DEVELOPMENT OF A REAL-TIME SMARTWATCH ALGORITHM FOR THE DETECTION OF GENERALIZED TONIC-CLONIC SEIZURES

by

Samyak Shah

A thesis submitted to Johns Hopkins University in conformity with the requirements for the degree of Master of Science in Engineering

Baltimore, Maryland

May 2019

Abstract

Generalized Tonic Clonic Seizure (GTCS) detection has been an ongoing problem in the healthcare industry. Algorithms and devices for this problem do exist on the market, but they either have poor False Positive Rates, are expensive, or cannot be used as anything other than a seizure detector. There is currently a need to provide a portable seizure detection algorithm that can meets patient demands. In this thesis, we develop a two-stage end-to-end seizure detection algorithm that is implemented on an Apple Watch, and validated on Epilepsy Monitoring Unit (EMU) patients. 124 features are extracted from the collected dataset, after which 9 are empirically selected. We have provided mutual information based feature selection methods that cannot yet be implemented on the watch due to computational restrictions. In stage one we compare common anomaly detection methods of One Class SVM, SVDD, Isolation Forest and Extended Isolation Forest over a thorough cross-validation to determine which is ideal to use as an anomaly detector. Isolation Forest (Sensitivity: 0.9, FPR: 3.4/day, Latency: 69s) was chosen despite the good sensitivity and latency of SVDD (Sensitivity: 1.0, FPR: 17.28/day, Latency: 8.9s) due to better implementation characteristics. During in-vivo testing, we record a sensitivity of 100% over 24 recorded tonic seizures with FPR: 1.29/day. To further limit false positive detections, a second stage is incorporated to separate between true and false positives using deep learning methods. We compare a Deep-LSTM, CNN-LSTM and TCN network. CNN-LSTM (Sensitivity: 0.93, FPR: 0.047/day) was finally used on the watch due to its tractable implementation, though TCN (Sensitivity: 1.0, FPR: 0/day) performed significantly better during cross-validation. During in-vivo testing, the 2-stage algorithm showed sensitivity: 100%, FPR: 0.05/day over 2004 tracked hours and 12 seizures. The mean latency was 62 seconds, which is on the threshold of clinical acceptability for this task.

Primary Reader: Dr. Nathan Crone

Secondary Readers: Dr. Gene Fridman, Dr. Shinji Watanabe

Acknowledgements

The following people had a significant influence on this thesis: Dr. Nathan Crone, Mr. Michael Chan, Mr. Maxwell Collard, Mr. Manar Alhamdy, Mr. Erie Gonzalez Gutierrez and Dr. Gregory Krauss.

Additionally I would like to thank my thesis committee members Dr. Shinji Watanabe and Dr. Gene Fridman for taking the time to read through and provide comments on this work.

I would also like to thank the nursing staff at Johns Hopkins University for the tireless work they have done in data collection, the developers at the Technology Innovation Center that developed the iOS architecture, and all the undergraduate students and interns that have contributed in the development of this algorithm.

Finally I would like to thank my family and friends for giving me an unending supply of support and inspiration.

Contents

List of Figures

List of Tables

Introduction

Background

Epilepsy is a neurological disorder that actively affects 1.2% of the US population [1]and is characterized by a paroxysmal alteration of neurological function due to abnormal and excessive synchronous brain activity known as an "epileptic seizure"[2]. Though epilepsy will generally present with some form of seizure, seizures are not always indicative of epilepsy. There are multiple subtypes of epileptic seizures, usually classified by their clinical EEG characteristics [3]. The original classification scheme of seizure types was developed in 1981, and was used for almost two decades. However that list was built on concepts that no longer correspond to or accurately describe modern knowledge of seizures and epilepsy [4]. The current classification system is shown in Figure 1 [5]. An expanded view is provided in the appendix.

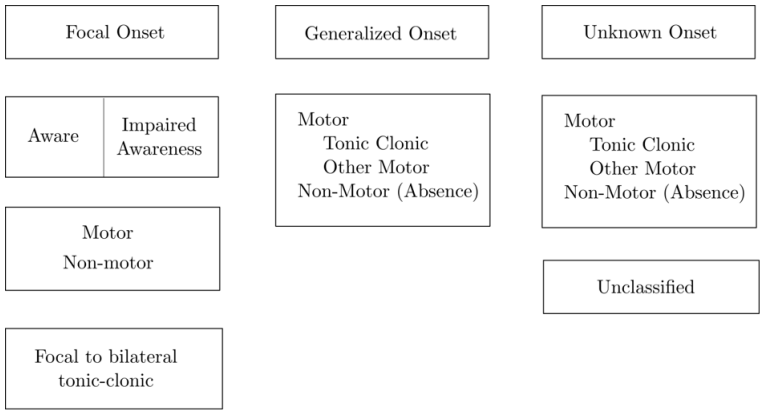


Figure 1: Simplified seizure sub-type diagram

All seizures are caused by abnormal electrical activity in the brain, with two main subtypes, focal and generalized. Focal seizures occur when the abnormal electrical activity in the brain is localized to a limited region. Generalized seizures will affect both cerebral hemispheres simultaneously at onset. Focal seizures may sometimes generalize, which we define as secondary generalizing seizures. Other seizures to note

for this thesis are myoclonic (brief shock like jerks of a muscle or a group of muscles), hypermotor (characterized by complex high amplitude movements of proximal segments of the body [6]), and focal aware/unaware seizures (non-motor partial seizures will freeze the body for up to two minutes).

All seizures present with a varying level of altered consciousness When the abnormal activity involves cortical and sub-cortical structures, as is the case with generalized tonic-clonic and absence seizures, loss of consciousness can also occur [7]. Generalized tonic-clonic seizures (GTCS) in particular (characterized by a stiffening in the tonic phase and a rhythmic jerking in the clonic phase) may lead to injuries, emotional distress, and reduced quality of life. GTCS are also thought to be an indicative risk factor of Sudden Unexpected Death in Epilepsy (SUDEP), especially if the patient is left unattended [8]–[10]. Due to these difficulties, patient demands [11]–[14] and seizure under-reporting (particularly nighttime seizures) [14], [15], there have been numerous seizure prediction/detection methods using both EEG and non-EEG based modalities [9], [16]–[19].

Intracranial and Scalp EEG based detection is the gold standard for seizure detection, but have the disadvantages of having being uncomfortable/obtrusive and invasive respectively. Other modalities like surface EMG (sEMG), electrodermal activity (EDA), electrocardiogram (EKG) and accelerometer (ACM) have shown promising results [17]. Due to the wide availability in common smartwatches ACM, heart rate and gyroscope signals are attractive modality choices. Despite the primary disadvantages of being limited to only detecting **unhindered** motor based seizures, they have been shown to effectively detect **GTCS, secondary generalized, myoclonic, clonic, tonic** and **hypermotor subtypes**, achieving sensitivities of 87.5% - 100% for GTCS at latencies ranging from 9-60s after clinical seizure onset. False positives are still high with only one system reaching a false-positive rate of 0.2/day [17]. Both patients and physicians require a maximum acceptable false positive rate of 0.14/day (once per week), and an idealized false positive rate of 0.03/day (once per month) according to a comprehensive survey performed

on seizure detection systems [20]. Additionally, most systems are expensive, and have no secondary purpose.

There is currently a need for a seizure detector high sensitivity and significantly higher specificity that can be implemented in a commercial device.

Scope

This thesis covers the end-to-end development of the seizure detector, including preprocessing, feature selection, model selection, cross-validation, implementation and testing. We will cover theory of the selected models and attempt to prove mathematical justifications where necessary. Some more technical proofs are omitted, but will be referenced. The focus of this thesis is on the algorithm for seizure detection, not the implementation on the Apple Watch, and these sections are summarized for brevity.

Aim

The aim of this thesis is to develop a real-time watch-based, generalized tonic-clonic seizure detection system that can be deployed in a commercial smartwatch with state-of-the-art results.

Data Collection and Storage

Users

There are two versions of the EpiWatch application, one with the detector implemented and one without. The detector version is used in the EMU and Beta users. The non-detector version is used by the general public. Both are able to record data, but in this thesis we shall only use data collected from either the EMU or Beta users (non-epileptic). This is because we can verify the data we collect from these sources, a task not possible with the general public.

We designate the users into 3 groups, corresponding to how our detector was developed. The groups are not independent.

Table 1: Patient demographic breakdown for research phases

	Total Users	EMU	Beta
Anomaly Detector Training and Validation	62	58	4
Secondary Detector Training	56	51	5
Secondary Detector Validation	36	30	6

Sensors and Collection Pipeline

We have made use of several iterations of the Apple Watch throughout the lifetime of the EpiWatch project. All of the watch versions (Series 1 – Series 4) have had a similar biosensor array, consisting of a Photoplethysomgraphy (PPG) sensor and a triaxial accelerometer. There is a gyroscope sensor available, however at present a public API does not exist. Other sensors (EKG, microphone, touch) are either too new, or not necessary for our current algorithm.

The accelerometer data was sampled at the Nyquist frequency of 50Hz (No seizure should cause vibrations in the tonic phase faster than 25Hz), and the heart rate data was calculated from the PPG sensor approximately every five seconds. This calculation is done one the watch by proprietary Apple software, so heart rate data can be sampled directly through the API. The data is stored on the watch in overlapping 1 minute segments, and periodically uploaded to our custom cloud storage unit. Each data point has an associated time stamp at storage so it possible to align the heart rate and accelerometer samples during detection and retrospective analysis. The overlapping data is necessary so that none of the data is accidently lost if there is any delay during data upload, and typically ranges around 10 seconds. Any data multiples in the overlap window are handled during preprocessing.

It is noted that in early versions of the Apple Watch (up until Series 3), there have been several issues with missing accelerometer and heart rate data. This missing data could last from a few seconds to several minutes, causing data gaps in recordings of both 'normal' activity and seizures. This was thought to be a software issue, and has mostly been alleviated with the release of the Series 4. Occasional gaps in the heart rate data coinciding with the tonic phase have still been observed, though this has likely been caused by improper contact of the sensor to the wrist. Additionally, as some asymmetric seizures will generalize to only side of the body, we would occasionally have seizure data with no valuable information. Seizures like this were not included in this study.

Originally, data was stored as JSON files, referenced through a NoSQL (MongoDB) database hosted on a local server. After the first 4 seizures, we updated to storing data as S3 binaries, referenced through a Postgres SQL database. Each watch running the EpiWatch app has a unique identification number through which data can be referenced. Data collected from users in the Epilepsy Monitoring Unit is stored in a separate database from the general population in order to maintain data integrity, with each seizure being cross-referenced by an epileptologist against a Video-EEG to determine validity, and create a dataset of ground truth seizures.

The databases also store a plethora of relevant clinical information that are out of the scope of this project. While most of these features are not directly relevant to a seizure detection algorithm at this time, some may inform interesting trends in the long term that allow for a more personalized detector. Detection information is also stored in the backend, though as of now, detection is only being offered to EMU patients and Beta users the app. It is enabled through a tracking option on the interface, will simultaneously alert both the user (through heavy vibrations and an alarm) and the primary caregiver (through text SMS) upon being triggered.

The user will also be presented with a prompt lasting 15 seconds requesting confirmation. If there is an affirmative or no response, this is followed by a clinically designed responsiveness test to measure awareness throughout the seizure. Responsiveness is unlikely during a primary GTCS, though is possible in secondary generalizing GTCS and other seizure subtypes. The detection time will be stored in the database under the unique watch ID.

This seizure detection routine can also be manually triggered by the user, and this data will be stored separately in the backend.

Preprocessing

Preprocessing is performed on the data before training to ensure consistency. This stage was necessary in both offline (retrospective) analysis, and real-time detection.

Offline Preprocessing Pipeline

The data was stored in 1-minute segments with overlaps to avoid any losses. If there was a value conflict on the same time sample, the second value was disregarded. Due to the data gaps that were possibly present in the data, an initial sorting step was also necessary to ensure the pulled data was mostly contiguous.

A loop was run through all the stored data. Any samples with time stamps separated by more than 100 milliseconds were cut and separated, leading to a set of shorter contiguous segments. Any segment less than 10 seconds in total length was discarded. 10 seconds is chosen as the threshold because some features will be calculated over a sliding window where the minimum length is 10 seconds. All heart rate samples are correspondingly grouped into the segments by time stamp. Each segment is saved independently according to a user and segment id.

Due to the different sampling rates between HR and accelerometer data, many accelerometer data points that do not have a corresponding HR value. During feature extraction, heart rate interpolation is performed by assigning the last available heart rate value to each point (zero-order hold interpolation scheme). In cases where no previous heart rate value is available, the mean value of 80 bpm is used.

This segmentation was not performed on the seizure data (this was excluded by cutting any segments contained in the seizure timestamps provided and verified by an epileptologist), as for semi-supervised

methods like Isolation Forest, Extended Isolation Forest, OCSVM and SVDD it was not necessary to use the seizure data for training. They would be needed during training, but we run the segments sequentially through the detectors to simulate real-time detection. While data gaps did exist in some seizures segments that could affect sensitivity and latency, they were rare and sporadic.

The accelerometer sample was then digitally high pass filtered with a cutoff frequency of 0.5Hz (2^{nd} order Butterworth) to remove the gravitational effect as well as any other low frequency trends. A low pass filter with a cutoff frequency of 20Hz (4^{th} order Butterworth) was also used to any remove high frequency noise and spiking artefacts. This filter was IIR (Transposed-Direct-Form II Structure), applied in one direction. While our application is pseudo-real time, this is a causal filter, and can be extended to real-time applications.

A final zero order hold interpolation was performed on the accelerometer data to ensure a uniform 50Hz sampling rate. This uniformization is necessary for extraction of any spectral features, and the data may still contain points that are too close together due to overlaps or data gaps below 100 milliseconds.

Real-Time Preprocessing Pipeline

On the watch, preprocessing is all completed in pseudo-real time. Data is sampled from the sensors into two buffer arrays for accelerometer and heart rate respectively. These arrays are grouped into 1 second blocks. Every 5 seconds, the collected blocks are processed by the detection algorithm. Each block is first filtered (with the same filter coefficients as in the retrospective method), and then interpolated using a zero-order hold scheme.

Then feature extraction is performed. Note that in the retrospective case, window features are calculated on a sliding window of 10 seconds. In real-time, 5 second payloads are passed to the detector, all block

features are extracted exclusively from the 1 second blocks. A dynamically updating circular buffer of 10 seconds is created to hold data for windowed feature extraction. They are updated by popping old data once the buffer length passes 500 samples (corresponding to 10 seconds at 50 Hz sampling rate), and pushing any incoming samples.

Heart rate features that require windowed data are also implemented using a similar buffering approach. As HR has a lower sampling rate, an equal buffer size corresponds to samples much further back in time. Thus windows for HR features can be much longer without causing memory concerns.

Note that data-gaps cannot be handled in real-time. Any gaps will cause artefacts due to both filtering and interpolation. For windowed features, the last known 10 second interval will be used. If the data gaps are too big, they will collectively culminate in erroneous predictions from the detector.

Features

Feature Extraction

A number of possible features were developed to extract from the data in the offline. Features were calculated from either a window, or a 1 second block. Motivation for selecting these features was either from use cases in time-series tasks (Activity Recognition, EEG-based seizure detection, Quantitative Finance), and intuition. These features are only implemented in offline pipeline. Only a subset will be implemented on the watch.

Heart Rate Features

Current Heart Rate

A low-cost time series feature to implement that is extremely telling. Most GTCS will have a significant increase in instantaneous heart rate (to between 140 and 180 bpm) for a short period of time that corresponds to the tonic and clonic phases.

Mean Heart Rate Difference

Heart rate changes in magnitude happen in the matter of seconds. The heart rate derivative feature captures the weighted mean of the heart rate derivative in a 30 second window. The weighted mean is to account for sampling inconsistencies.

$$\text{HR Mean} = \frac{1}{T}\sum_{i} \Delta t_i \Delta HR_i$$

Where $T = \sum_i \Delta t_i$, ΔHR_i is the difference between any two consecutive HR samples, and Δt_i is the corresponding difference in time.

Median Heart Rate Difference

This is a custom feature built on comparing the median heart rate between two long long, non-overlapping windows. It gives a more stable insight on heart rate changes, and is a robust solution to data gaps. Start with two user defined parameters of far-window length t_f and near-window length t_n in seconds, with the current time defined by t. This feature is calculated by

$$\text{HR Median}(t_s, t_f) = \text{median}(\text{HR samples from } t - t_s \text{ to } t) - \text{median}(\text{HR samples from } t - t_f \text{ to } t - t_s)$$

Heart Rate Latency

During GTCS, we often witness heart rate data drops in the high activity regions. This is likely caused by lack of consistent contact between the PPG sensor and the skin. Knowing that they can occur, we can use these data drops as a potential feature called heart rate latency. We calculate heart rate latency is a weighted average of the time difference between heart rate samples.

Temporal Features

Mean L2 – Norm

Also known as the Euclidean norm of the signal. It is easy to implement in a real-time environment, and gives an idea of the total energy of the signal. The mean L2 should markedly increase during GTCS. Given accelerometer data matrix $\boldsymbol{A} \in \mathbb{R}^{N\times 3}$, where a single accelerometer sample is represented by column i, $\boldsymbol{A}_i = [A_{i1}, A_{i2}, A_{i3}]$, and the columns stands for the x, y, z samples, the mean L2 norm is given by

$$\text{Mean} = \frac{1}{N}\sum_i \left(\sum_j A_{ij}^2\right)^{\frac{1}{2}}$$

Line Crossing Rate

This feature corresponds to the total count of sign changes signal accumulates within a certain period of time. Alternatively it is thought of as the amount of times the signal has crossed the 0 line. It is a simplified measure of the frequency of a signal. GTCS have a characteristic frequency pattern starting at 8 Hz in the tonic phase before slowing to about 1.5 Hz in the clonic phase [21]. This produces a characteristic descending frequency chirp that may be recognized by temporal classifiers using this feature. Taking $\boldsymbol{A} \in \mathbb{R}^{N\times 3}$ as an accelerometer data matrix, LC rate can be formulated for one axis as

$$\mathrm{LC} = \frac{1}{2T}\sum_{i=2}^{N}\left|sign\left(A_{ij}\right) - sign\left(A_{(i-1)j}\right)\right|$$

Where T is the window size. This gives the line crossings for one of the accelerometer directions **per unit of time**. LC rates for the orthogonal axes must be calculated independently. We halve the values because we accumulated line crossings count is implicitly doubled when using the sign() function. We also set a threshold on the line crossing amplitude to mitigate effects of low amplitude noise.

Mean Line Crossing Rate Derivative

This is a filtered measure of the derivative of the line-crossing rate, often referred to as the velocity or divergence of the signal. Mean LC Rate a metric commonly used in quantitative finance, calculated by computing the difference between the short term and long-term exponential moving average (EMA) of a signal. A flat signal will have a low divergence, but a stable long term signal with jittery short term characteristics (tonic phase of a seizure after the segment has been filtered to remove any drift) will have have a correspondingly high divergence. This feature is also calculated individually for each axis in the accelerometer. Taking the accelerometer data matrix as $\boldsymbol{A} \in \mathbb{R}^{N\times 3}$

$$\text{LCD} = \frac{1}{N-1}\sum_{i=2}^{N}\left[\alpha_s A_{ij} + (1-\alpha_s)A_{(i-1)j}\right] - \left[\alpha_l A_{ij} + (1-\alpha_l)A_{(i-1)j}\right]$$

where α_s is the coefficient for the short term EMA, and α_l is the coefficient for long term EMA. Note we eventually take the average over the window size.

Percentile

This feature returns the value of the data at the n^{th} percentile. It is an easy feature to implement, and can give an idea of the amplitude distribution. In a high pass filtered signal (centered), 50th percentile will almost always return 0 during seizure segments. The edge percentiles (i.e 90th or 10th) for the same segment will return higher/lower values.

Variance (Hjorth Activity)

Variance provides a statistical method of measuring the variation from the mean in the data. Defined on a signal it is also known as Hjorth activity, and represents the signal power (0th spectral moment). It indicates the surface of the power spectrum in the frequency domain [22]. Seizures will generally provide high Hjorth activity in both tonic and clonic phases. Letting $\boldsymbol{A} \in \mathbb{R}^{N\times 3}$ represent the accelerometer data matrix, and $\boldsymbol{\mu} \in \mathbb{R}^{3\times 1}$ represent the corresponding axis means, the variance along any individual accelerometer axis is calculated by

$$\text{Var} = \frac{\sum_i \left(A_{ij} - \mu_j\right)^2}{N}$$

Standard Deviation

This feature Similar to variance in that it provides a metric of dispersion in the signal. Unlike variance, it has the advantage of being defined in the units of the variable we are observing. It is calculated by taking the square root of the variance.

Normalized Jerk

This feature is the normalized rate of change of acceleration, implemented to capture the direction changes and high acceleration that is present during GTCS, particularly during the tonic phase. We believe this may help with distinguishing seizures from similar rhythmic activities like running. Knowing accelerometer readings return acceleration data, and taking the acceleration data matrix as $\boldsymbol{A} \in \mathbb{R}^{N\times 3}$,

$$\text{Jerk} = \frac{1}{N}\left(\sum_{i}^{N}\left(\frac{A_{ij} - A_{(i-1)j}}{\Delta t_i}\right)^2\right)^{\frac{1}{2}}$$

where Δt_i represents the time difference between the sample A_{ij} and $A_{(i-1)j}$.

Hjorth Mobility

Mobility can be interpreted as the standard deviation of the power spectrum of a signal along the frequency axis. It is also known as the 2nd spectral moment [22]. To calculate, we take $\boldsymbol{A} \in \mathbb{R}^{N\times 3}$ to be the accelerometer data matrix, with $\boldsymbol{A}_{.j}$ defining the j^{th} column, and $\text{Var}(\cdot)$ as a function that calculates the variance. $\boldsymbol{A}'_{.j}$ is the **discrete derivative** of the vector $\boldsymbol{A}_{.j}$, calculated as $\frac{A_{ij}-A_{(i-1)j}}{\Delta t_i}$.

$$\text{Mobility} = \sqrt{\frac{\text{Var}\left(\boldsymbol{A}'_{.j}\right)}{\text{Var}\left(\boldsymbol{A}_{.j}\right)}}$$

Hjorth Complexity

Complexity (4^{th} spectral moment) is a dimensionless parameter, signifying the similarity of a signal to a pure sine wave [22]. This feature was created due to the oscillatory nature of characteristic GTCS. A variety real time tasks like walking, running and brushing of teeth will carry a similar oscillatory signal, potentially leading to more false positives that will have to be standard. Taking $\text{Mobility}(\cdot)$ as a function that calculates mobility,

$$\text{Complexity} = \frac{\text{Mobility}(\boldsymbol{A}'_{\boldsymbol{j}})}{\text{Mobility}(\boldsymbol{A}_{\boldsymbol{j}})}$$

Note all the Hjorth parameters calculate spectral statistics in the time-domain, and are a low-cost alternative to calculating specific spectral features through explicitly defining the power spectral density matrix.

Root Mean Square Energy

Also known as the quadratic mean, it is defined as the square root of the arithmetic mean of the squared signal values. It is a commonly used statistic in EEG feature extraction [23], as well as in electrical engineering. It gives a sense of the absolute magnitude of the average value of a signal. One again, we calculate it individually for each axis

$$\text{RMS} = \left(\frac{\sum_i (A_{ij})^2}{N}\right)^{\frac{1}{2}}$$

Line Length

Defined as the running sum of absolute differences between consecutive samples in a predefined window. This feature has been used successfully in EEG based seizure detection [24], and is an approximation of

the fractal dimension of a signal [25]. This makes it efficient in detecting signal transients, motivating our use case.

$$\text{Line Length} = \sum_{i}^{N} \left| A_{ij} - A_{(i-1)j} \right|$$

Magnitude Area

This feature gives an estimate on the area under the signal envelope. It is a commonly used feature with accelerometer data, and has been used previously for activity recognition tasks [26]. While good for seizure detection, it will also capture other vigorous activity leading to additional false positives.

$$\text{SMA} = \sum_{i} |A_{ij}|$$

Energy

This feature is similar to magnitude area. Is defined as the area under the squared magnitude of the signal. Due to the squared term inside the sum, higher magnitudes will be amplified compared to lower magnitudes. It is a common feature in signal processing.

$$\text{Energy} = \sum_{i} A_{ij}^2$$

Normalized Energy

Due to imperfect sampling of our seizure, we use a normalized energy to allow comparison of signals with varying lengths.

$$\text{Normalized Energy} = \frac{1}{N}\sum_{i} A_{ij}^2$$

Skewness

Skewness is a higher order statistical feature which represents the symmetry of the signal amplidude probability density function (PDF). It is also known as the 3rd standardized moment. A perfectly symmetrical function will have skewness 0. Any time series with a few small values and many large values (left tail) will have negative skewness, while many small values and a few large values (right tail) will have positive skewness. We use standardized moment for scale invariance. Taking μ_j, σ_j as the arithmetic mean and standard deviation of $\boldsymbol{A}_{:j}$ respectively

$$\text{Skewness} = E\left[\left(\frac{\boldsymbol{A}_{:j} - \mu_j}{\sigma_j}\right)^3\right] = \frac{1}{N\sigma_j^3}\sum_{i}^{N}\left(\boldsymbol{A}_{:j} - \mu_j\right)^3$$

Kurtosis

This is a higher order statistical feature which represents the 'peakedness' of the signal amplitude PDF. It is also known as the 4th standardized moment. A kurtosis value close to **three** will indicate Gaussian-like peakedness. Sharper peaks will correspond to higher kurtosis values. Taking μ_j, σ_j as the arithmetic mean and standard deviation of $\boldsymbol{A}_{:j}$ respectively

$$\text{Kurtosis} = E\left[\left(\frac{\boldsymbol{A}_{:j} - \mu_j}{\sigma_j}\right)^4\right] = \frac{1}{N\sigma_j^4}\sum_{i}^{N}\left(\boldsymbol{A}_{:j} - \mu_j\right)^4$$

Spectral Features

All spectral features are derived from a power spectral density (PSD) of a signal, estimated using Welch's method. The PSD shows the power of the signal at varying frequencies. We use Welch's method as it mitigates noise estimations in the frequency domain, and it is generally a good non-parametric approach that can be employed as a baseline. In formulas we represent it by the vector $\boldsymbol{P} \in \mathbb{R}^F$, where F is the number of frequencies in the PSD.

Dominant Frequency

This feature finds the frequency value corresponding highest power in the PSD. Oscillation frequencies during tonic and clonic phases are well characterized during GTCS, and contain a descending chirp which may help temporal classifiers correctly determine whether a segment of data is a seizure or not. The dominant frequency may provide information in this regard.

Spectral Edge Frequency

A popular feature in EEG monitoring [27], comparable to the percentile measurement in the time domain. SEF determines the frequency below which x percent of the total signal power is located. To calculate, we can run a running sum on PSD vector $\boldsymbol{P} \in \mathbb{R}^F$, until we reach the required spectral edge value $\alpha \in [0,1]$. The frequency we stop on is the SEF

$$\text{SEF} = f_s,$$

where f_s is maximum frequency value that satisfies

$$\sum_{i=0}^{f_s} P_i \leq \alpha$$

Spectral Band Power

Measures the power of a signal in a chosen frequency band. Band power is another popular feature in EEG monitoring. Given user input frequencies of f_l, f_h, where $f_l \leq f_h$

$$\text{SpectralBP} = \sum_{i=f_l}^{f_h} P_i$$

Spectral Centroid

This feature indicates the "center of mass" of the PSD. It is calculated by performing a weighted sum over all frequency values in $\boldsymbol{P}$. Note when calculating the spectral centroid, i defines individual frequency values, with P_i being the corresponding power.

$$\text{Spectral Centroid} = \sum_i \frac{iP_i}{\sum_j P_j}$$

Spectral Entropy

Calculates the complexity of a signal by taking the entropy over its PDF. Entropy is an information theoretical concept that determines the uncertainty in some stochastic source. White noise will have highest spectral entropy, while all the power being focused on a single frequency will have a spectral entropy of 0. We will cover entropy in more detail during the feature selection section. To determine the PDF of $\boldsymbol{P}$, we normalize. Defining $\boldsymbol{P_n} \in \mathbb{R}^F$ as the normalized PSD, and setting it elementwise by $P_{n_i} = \frac{P_i}{\sum_j P_j}$

$$\text{Spectral Entropy} = -\sum_i P_{n_i} \ln P_{n_i}$$

We calculate 124 features in total, all derived from this base feature set. The feature names are given in table 2. Some features were additionally smoothed through exponential average filters, and the corresponding α −parameter of these filters is provided in the title. Any features without an asterisk (*) must be calculated once for each accelerometer axis

Table 2: List of all extracted features for offline analysis

Temporal Acc	Spectral Acc	Heart Rate
L2 Norm*	Dominant Frequency	Current Heart Rate*
Percentile (25)	Band Power 0-20 Hz	Heart Rate Latency*
Percentile (50)	Band Power 0-2 Hz	Mean HR Derivative*
Percentile (75)	Band Power 2-4 Hz	Delta Median (30, 120)*
Jerk	Band Power 4-6 Hz	Delta Median (60, 120)*
Variance	Band Power 6-8 Hz	Delta Median (60, 180)*
Standard Deviation	Band Power 8-10 Hz	
Mobility	Band Power 10-12 Hz	
Complexity	Band Power 12-14 Hz	
RMS	Band Power 14-16 Hz	
Line Length	Band Power 16-18 Hz	
Magnitude Area	Band Power 18-20 Hz	
Energy	Spectral Edge (0.1)	
Normalized Energy	Spectral Edge (0.5)	
Skewness	Spectral Edge (0.85)	
Kurtosis	Spectral Edge (0.9)	
LC Rate (LCR)	Spectral Edge (0.95)	
LC Rate Derivative (LCRD)	Spectral Centroid	
Smoothed LCRD $\alpha = 0.02$	Spectral Entropy	
Smoothed LCRD $\alpha = 0.002$		
Smoothed LCRD $\alpha = 0.0002$		

Feature Selection

During original algorithm implementation, features were selected by hand. We visualized normal and seizure data while visualizing corresponding feature activations. Features that appeared most informative were selected. Retrospectively it was decided to perform an information theory based feature selection process for future algorithm iterations. Two such methods are described in this section.

Naïve Mutual Information (MIFS)

While the goal of classifiers is to best approximate a function to accurately predict labels of novel patterns, the limited amount of data will often cause the classifier to overfit to the data in practice. Additionally, a large number of features will significantly slow down the learning process. By judiciously selecting only the relevant features, we can both reduce overfitting, and increase computational speed of both training and classification.

Most of the existing feature selection algorithms can be separated into two methods, filter [28]–[30] and wrapper [28], [31]. Filter methods will select features independently from any learning algorithm, using statistics derived from the training data like distance, information and consistency. Wrapper methods use an exhaustive approach with a predetermined classifier to evaluate performance of varying subsets of features. This often leads to superior performance, at the expense of efficiency. Due to computational restrictions, we have decided to use a filter method for our feature selection, with mutual information being the selection statistic derived from the dataset.

In general, classifiers can be considered to be systems that use information in the input data to remove uncertainty of output class selection. While some classifiers perform remarkably well, all real-world implementations will have some form of residual uncertainty, stemming from either **insufficien**t or **inefficient** data. Of the two, inefficient data is easier to remedy, and can be done by choosing either more features (with the trade-off of higher complexity) or more informative features. To make sure all the data we are using is efficient, we take a further look at the concept of uncertainty.

'Uncertainty' can be quantified by an information theoretic concept called entropy. If C is the set of classes $\{c_1, \dots, c_N\}$, and $P(c_i)$ is the prior probability for each class, entropy is calculated by

$$H(C) = -\sum_{i=1}^{N} P(c_i)\, log\, P(c_i)$$

When given a set of feature vectors F with M individual feature vectors f we can define conditional entropy as

$$H(C|F) = -\sum_{j=1}^{M} P(f)\left(\sum_{i=1}^{N} P(c_i|f_j)\, log\, P(c_i|f_j\right)$$

Here, $P(c|f)$ is the conditional probability of a class given an input vector. In the case of continuous variables, the sum will be replaced with an integral. In general, conditional entropy will be lower than the initial entropy as we are providing additional information. Conditional entropy will be equal to initial entropy only when there is **general independence** between feature and output class, $P(c,f) = P(c)P(f)$. We define mutual information $I(C;F)$ as the amount by which uncertainty is decreased by adding the extra information. Note that it is a symmetric metric.

$$I(C;F) = I(F;C) = H(C) - H(C|F) = H(F) - H(F|C)$$

$$I(C;F) = I(F;C) = \sum_{C}\sum_{F} P(c,f)\, log\, \frac{P(c,f)}{P(c)P(f)}$$

Note that this form is similar to the Kullback Liebler (KL) divergence, and indeed can be written as $I(C;F) = D_{KL}(P(c,f)\ ||\ P(c) \otimes P(f))$, where $\otimes$ is the tensor product. We informally write this quantity as $D_{KL}(P(c,f)\ ||\ P(c)P(f))$. Mutual Information measures the "bumpiness" of the joint distribution. Qualitatively, joint probability functions that are flat in the limit will tend to 0 for mutual information, while "bumpier" joint probabilities (indicating higher general correlation) will have higher mutual information. If both C and F were independent, then $P(c,f) = P(c)P(f)$, and thus $I(C;F) = 0$.

The motivation for choosing mutual information as the similarity metric was due to its capability of measuring a general dependence (both linear and non-linear) between two variables. As an example, consider an XOR function of two input variables with equal probabilities for possible inputs. The correlation between any one of the two input variables and the output variable will be 0, as $Corr \propto Cov(X,Y) = E(XY) - E(X)E(Y) = 0$, because $E(XY) = E(X)E(Y)$ [See Appendix]. However, when calculating the mutual information between the input vector and the output, we are left with 1 bit. In other words, the input vector determines the output variable with no ambiguity. Though X_1 is pairwise independent of Y and X_2 is pairwise independent of Y, the vector (X_1, X_2) still uniquely determines Y. General independence implies linear independence, but not vice-versa, and MI is a measure of general dependence, especially useful for non-trivial probability densities [32].

After initial preprocessing, we have 124 computed features, which is infeasible to deploy on hardware (as well as adding unnecessary complexity). Given the set of features F, we want to select a subset S, $|S| < |F|$, wherein the selected features are maximally informative about the class.

Calculating the mutual information for every possible feature vector is computationally impractical, forcing us to consider approximate solutions, like the mutual information based feature selection (MIFS) algorithm [32].

MIFS works by calculating MI with only individual features, like $I(f; c)$ and $I(f; f')$, instead of with the feature vectors as a feasible approximation. The algorithm takes a greedy approach to feature selection. Given a set of selected features, it selects the next best feature based on maximizing the MI with the class variable, and then minimizing the average MI of the new feature when compared to the already selected feature set. The motivation here is to not pick dependent features, even though they may give good class

information. A β parameter is chosen to regulate the importance of this redundancy penalizing term, leading to the loss function:

$$\mathcal{L}(f; C, S) = I(C; f) - \beta \sum_{s \in S} I(f; s)$$

Table 3: Algorithm for MIFS feature selection process

Algorithm 1: MIFS
1. Set $F \leftarrow initial\ feature\ set$ and $S \leftarrow \{\emptyset\}$, initialize k, β
2. **for** f in F:
3. Compute $I(C; f)$ and store
4. end
5. Identify feature f that maximizes $I(C; f)$
6. Set $F \leftarrow F \backslash \{f\}$, $S \leftarrow \{f\}$
7. **while** $\lvert S \rvert < k$:
8. **for** all features $f \in F$, $s \in S$:
9. compute $I(f; s)$ and store
10. **end**
11. Identify feature f that maximizes $I(C; f) - \beta \sum_{s \in S} I(f; s)$
12. Set $F \leftarrow F \backslash \{f\}$, $S \leftarrow \{f\}$
13. end
14. Output selected features set S

In practice, it was found that $\beta = 1$ is often optimal, though there is not any theoretical justification to back this claim [33].

Conditional Mutual Information (mMIFS-u)

A more sophisticated method of feature selection can be obtained by observing conditional mutual information. The definition of conditional mutual information, similar to the definition of conditional probabilities, is

$$I(C;\ f_i|f_s) = H(f_i|f_s) - H(f_i|C, f_s))$$

C represents the class variable, while the f's represent feature vectors. Conditional independence denotes the mutual information of two variables conditioned on the third, with the right-hand side following from the definition of conditionality (analogous to probability).

To develop a greedy feature selection method, we must find a computationally friendly method of calculating conditional mutual information $I(C; f_i|f_s)$ [28]. We begin by proving two propositions

Proposition 1: The conditional mutual information can be represented as

$I(C; f_i|f_s) = I(C; f_i) - [I(f_i; f_s) - I(f_i; f_s|C)]$.

Proof:

$$
\begin{aligned}
& I(C; f_i) - [I(f_i; f_s) - I(f_i; f_s|C)] \\
&= H(C) - H(C|f_i) - [H(f_i) - H(f_i|f_s)] + H(f_i|C) - H(f_i|f_s, C) \\
&= H(C) - H(C|f_i) - H(f_i) + H(f_i|f_s) + H(f_i|C) - H(f_i|f_s, C) \\
&= H(f_i|f_s) - H(f_i|f_s, C) + H(C) - H(C|f_i) - [H(f_i) - H(f_i|C)] \\
&= I(C; f_i) - I(C; f_i) + H(f_i|f_s) - H(f_i|f_s, C) \\
&= I(C; f_i|f_s)
\end{aligned}
$$

Proposition 2: The ratio of entropy of f_s and MI between f_s and f_i is not dependent on conditioning by the class variable.

$$\frac{H(f_s|C)}{I(f_i; f_s|C)} = \frac{H(f_s)}{I(f_i; f_s)}$$

Translated to seizure detection, this assumption would hold as no matter if the data is coming from a seizure or not, the ratio of entropy and mutual information will hold.

Using proposition 1 and 2

$$
\begin{aligned}
I(C; f_i|f_s) &= I(C; f_i) - [I(f_i; f_s) - I(f_i; f_s|C)] \\
&= I(C; f_i) - \left[I(f_i; f_s) - \frac{I(f_i; f_s)}{H(f_s)} H(f_s|C) \right] \\
&= I(C; f_i) - \left[\frac{I(f_i; f_s)}{H(f_s)} H(f_s) - \frac{I(f_i; f_s)}{H(f_s)} H(f_s|C) \right] \\
&= I(C; f_i) - \frac{I(f_i; f_s)}{H(f_s)} \big(H(f_s) - H(f_s|C) \big) \\
&= I(C; f_i) - \frac{I(f_i; f_s)}{H(f_s)} I(f_s; C)
\end{aligned}
$$

In this form, we see how conditional mutual information measures the information of each new feature relative to a class, whilst penalizing a weighted dependency term. To pick the best feature, the optimization would be

$$
f = \max_{f_i \in F \setminus S} \left\{ I(C, f_i) - \max_{f_s \in S} \frac{I(f_i; f_s)}{H(f_s)} I(C, f_s) \right\}
$$

where F is the initial feature set, and S is the feature subset. The form of this feature selection method is identical to the naïve case, except now the weighting parameter is automatically updated through the conditional mutual information.

Table 4: Algorithm for mMIFS-u feature selection process

Algorithm 2: mMIFS-U

1. Set $F \leftarrow initial\ feature\ set$ and $S \leftarrow \{\emptyset\}$; initialize k; initialize mi_class_store array
2. **for** f in F:
3. Compute $I(C; f)$ and append to mi_class_store
4. **end**
5. Identify feature f that maximizes $I(C; f)$
6. Set $F \leftarrow F \backslash \{f\}$, $S \leftarrow \{f\}$
7. Intialize entropy storage array H_store of size k
8. $H\_store[0] \leftarrow$ Calculate entropy $H(f)$
9. Initialize $k \times |F|$ matrix mi_feat_store
10. **while** $|S| < k$:
11. **for** ind_f, f in enumerate (all features $f \in F$):
12. $mi\_feat\_store[|S| - 1][ind\_f] \leftarrow I(f_i; f_{s_{new}})$, where $f_{s_{new}}$ is the latest selected feature
13. **end**
14. Initialize outer maximization array $outer\_arr$
15. **for** ind_f, f in enumerate(all features $f \in F$):
16. Initialize inner maximization array $inner\_arr$
17. **for** ind_s, s in enumerate(all features $s \in S$):
18. compute $\frac{mi\_feat\_store[ind\_s][ind\_f]}{H\_store[s]}\ mi\_class\_store[s]$ and append to $inner\_arr$
19. **end**
20. $inn\_max\_val \leftarrow$ Maximum value of $inner\_arr$
21. append $mi\_class\_store[ind\_f] - inner\_max\_val$ to $outer\_arr$
22. **end**
23. Identify index and corresponding feature f maximizing $outer\_arr$
24. Set $F \leftarrow F \backslash \{f\}$, $S \leftarrow \{f\}$
25. **end**
26. Output selected features set S

Increases in speed can be gained through parallelization, especially in feature sets with a higher cardinality. Selection of the first and second features are illustrated in Figures 2 and 3 The first feature is selected through the highest mutual information value against the class variables. This is Delta Median (30, 180). Next we calculate the self-entropy of all the feature. This array will be stored in cache and reused during each feature selection loop. Using the mutual information between class variables, features, and entropy values, we next calculate the weighted redundancy term for each feature with the Delta Median (30, 180). Finally, we use the derived formula to calculate the conditional mutual information for the entire feature set, showing us the best second feature is heart rate. We recursively perform these calculations until we reach our chosen feature subset cardinality.

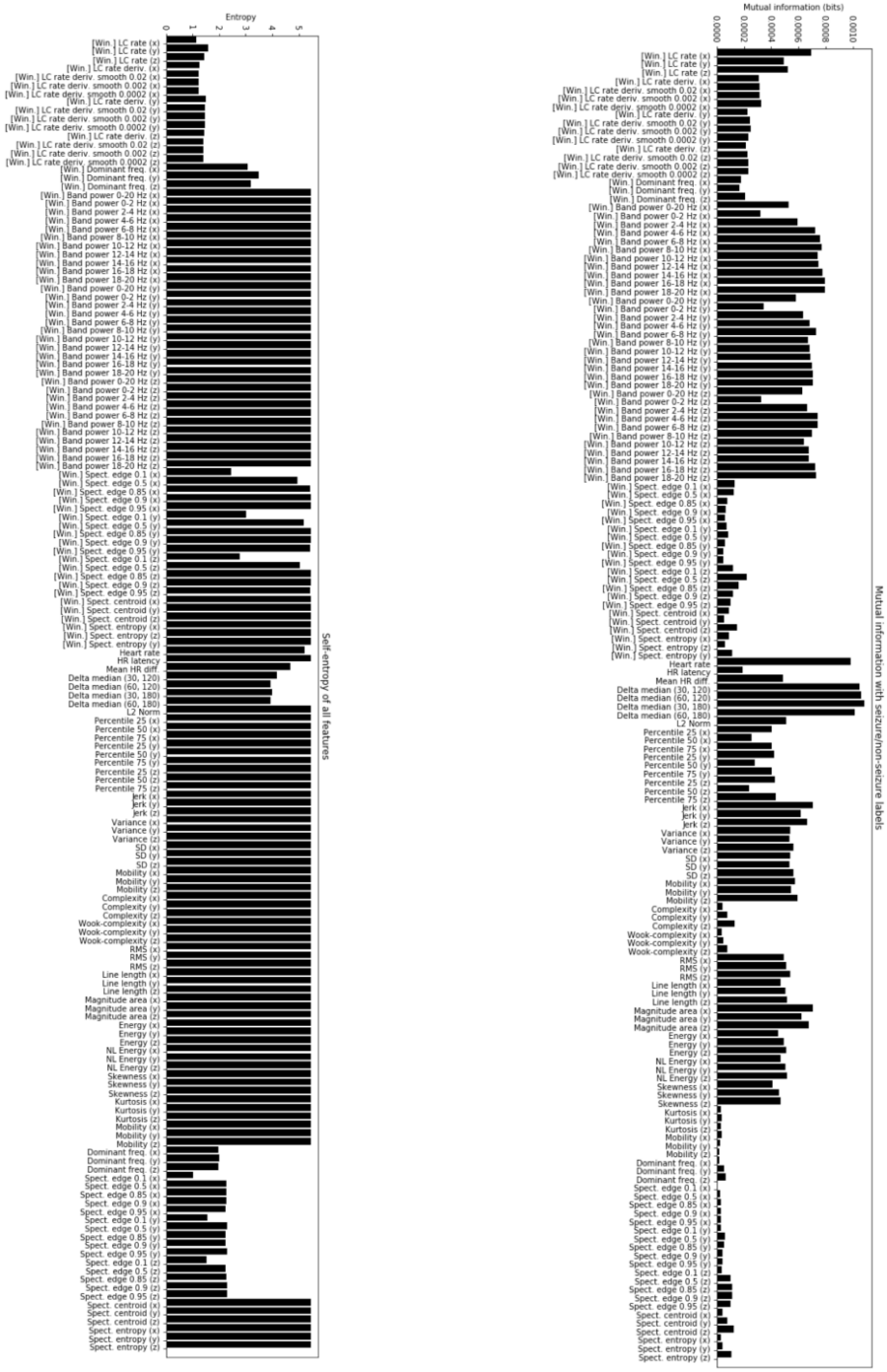


Figure 2: mMIFS-u feature selection algorithm applied in the context of seizure detection. a) shows the mutual information of all features with the class variable. b) shows the self-entropy of each feature

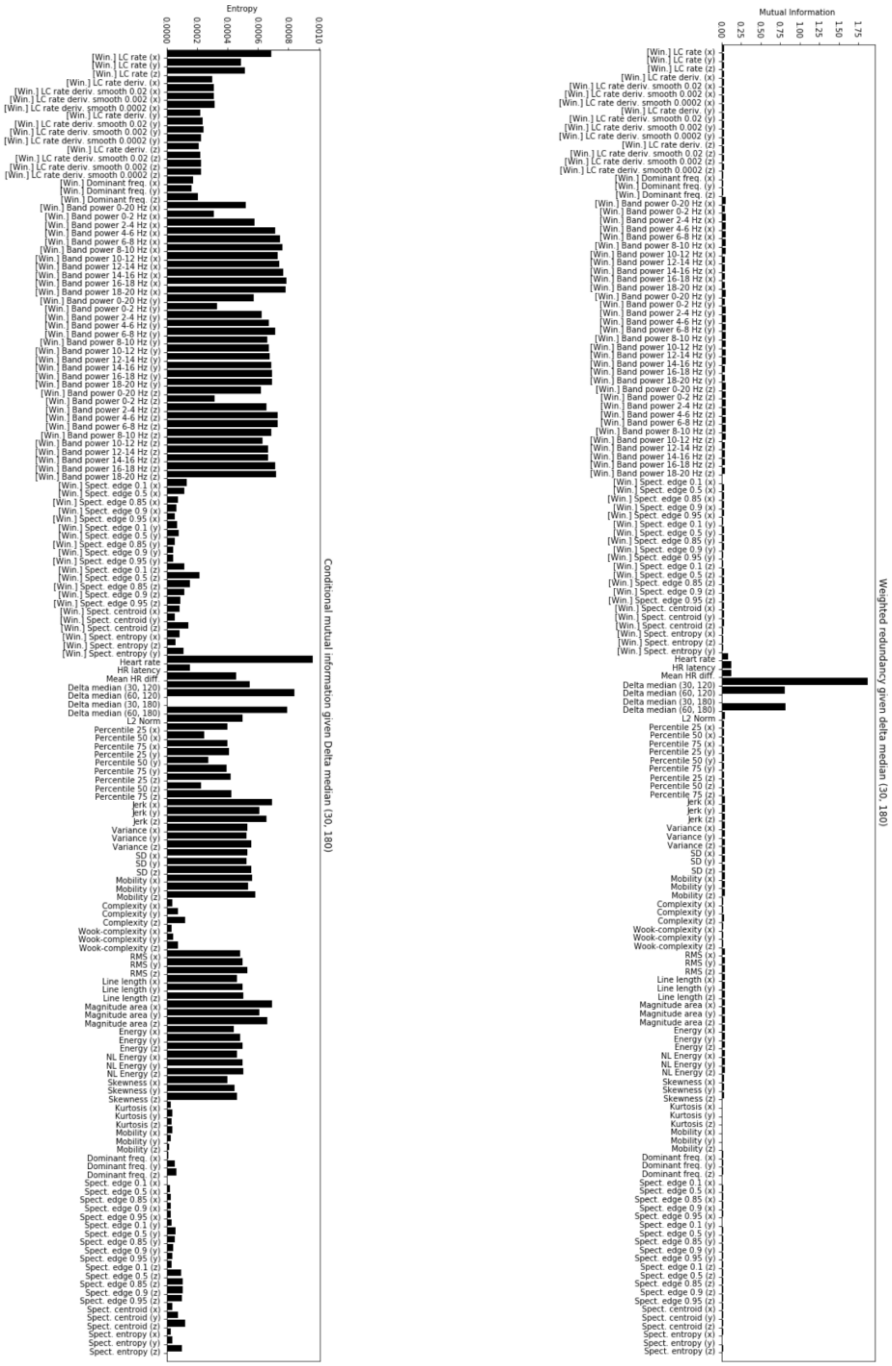


Figure 3: mMIFS-u feature selection algorithm applied in the context of seizure detection. a) shows the weighted redundancy term of all features compared to selected feature subset (Delta Median (30, 180)). b) shows final conditional entropy of all features

We selected 9 features using MIFS and mMIFS-u. These 9 features are shown from most informatic to least informatic according to the 2 algorithms. We choose $\beta = 0.001$ to prevent over-penalization of redundancy which was giving us too many uninformative features.

Selection Results

Table 5: MIFS and mMIFS-u feature selection results

MIFS ($\boldsymbol{\beta = 0.001}$)	mMIFS-u
Delta Median (30, 180)	Delta Median (30, 180)
Heart Rate	Heart Rate
[Win.] Band Power 18-20Hz	Delta median (60, 120)
[Win.] Line Crossing rate (x)	[Win.] Band Power 16-18Hz
Mean HR derivative	[Win.] Line Crossing rate (x)
Delta median (60, 120)	Jerk (x)
Kurtosis (y)	Delta median (30, 120)
Kurtosis (x)	Delta median(60, 180)
Kurtosis (z)	[Win.] Band Power 8-10Hz

Note that most of the features are not shared between the two algorithms. In the initial mutual information calculation, it is seen that the heart rate features are highly informative but are concurrently also highly dependent. Also note that the x-axis seems most informative. Despite this mMIFS-u still seems to favor heart rate features, especially delta median. Perhaps most interestingly, all three kurtosis values have been selected by MIFS. As kurtosis seems to be one of the most uninformative features according to Figure 2a, it stands to reason that it is highly independent from the other selected features and from kurtosis on other axes. To evaluate these features selection methods, the selected features would need to be tested on classification algorithms and compared.

At the beginning of this project no information theory based feature selection was performed. Instead features were chosen empirically by comparing trends on a custom-built visualization system. There were also system constraints limiting our choices on early watches. Due to the 15% CPU ceiling, it was not possible to calculate the PSD matrix in real-time without significantly hindering detection latency. This

ruled out any spectral features at that time, despite their high performance. With the release of a more powerful CPU in the Series 4, in addition to the availability of optimized libraries on iOS, spectral features will be available in the future iterations of the algorithm.

The following 9 features were chosen and implemented. Coincidentally, a few of the features correspond to those selected by MIFS, including HR, HR derivative and LC rate. A comparison of feature implementations between Swift (iOS) and Python (Offline) is provided.

Table 6: Selected Features Implementation Details

Feature	Offline Implementation	Real-Time Implementation
Current Heart Rate	Zero order hold interpolation. Mean is taken to calculate heart rate per second. In the case of no heart rate, 80 bpm average is used.	Causal nearest neighbors interpolation on the last available heart rate is used. In the case of multiple heart rates in the span of 1 second, the average is used. If no data is available, 0 is returned (will only occur before first sample)
Heart rate derivative	Computes a heart rate derivative estimate by finding the average derivative between samples within a 30 second sliding window. Due to the variation in sampling, mean value is weighted by Δt between HR samples. If less than 2 samples in a 30 second sliding window, 0 is returned.	Running circular buffer of 120 samples is implemented. Samples categorized by timestamp to be in the past 30 seconds is used for calculation. Same derivative and normalization as in the offline implementation. If less than 2 samples in the 30 second window, 0 is returned.
L2-Norm Accelerometer	Computes mean L2 Norm of the accelerometer data in each 1 second block. Gives an indication of the energy in the signal.	Computes mean of L2 Norm on each 1 second block.
X, Y, Z Line Crossing Rate	Computes LC rate on a sliding window size of 10 seconds. An empirically determined threshold of 0.05 has to be passed for a sign change to register as a line crossing to avoid spurious movements.	Compute LC on 10 second window as in offline implementation. 10 second window is implemented as circular buffer of 500 samples.
X, Y, Z Line Crossing Rate Derivative	Short term parameter $\alpha_s = 0.05$ and long-term parameter $\alpha_l = 0.005$. Difference is taken between the two EMA values, and the mean is taken over all values in the sliding 10 second window to find feature.	Compute the two moving averages on a 10 second window implemented as a circular buffer. Identical implementation to offline case.

Anomaly Detection

In development of this seizure detector, careful examination of our data and product constraints were performed to evaluate candidate models. As GTCS are a rare occurrence, the dataset was highly unbalanced. At the time of training, there were approximately 2000 total usable hours. Of this, there were ten validated GTCS, totaling 30 minutes. This sort of imbalance is commonly solved by anomaly detection algorithms, many of which have had successful applications in fields such as bank fraud, structural defects, and textual errors [34]. Outliers are categorized as either [35]:

1. **Global (Point Anomaly)**: Objects that deviate significantly from the rest of the data set, i.e. Meteors hitting earth
2. **Contextual Anomaly**: Objects that deviate significantly from the data set based on a selected context, i.e. Snow fall in the summer
3. **Collective Anoomaly:** A subset of objects collectively deviate significantly from the whole dataset, even if individual objects may not be outliers, i.e. DDoS attacks

Due to the rare and aggressive nature of GTCS, it is a global outlier, which leads to the challenge of finding an appropriate measurement of deviation from the standard dataset. This leads to the plethora of anomaly detection algorithms, which are also classified into three overarching types [34]:

- **Type I:** For datasets where there is no prior knowledge of the data, algorithms must determine a method of classifying the data according to some metric in feature space. Boundaries can be formed around groups, and defined as normal or anomalous. Test points are classified as outliers if they are not inside the normal regions. This is an unsupervised learning mechanism and most algorithms (generally clustering) assume anomalies and normal data have some distinct separation in feature space.

- **Type II:** If we have a dataset that is labelled, we can create a classifier that will group all the points according to their labels. This is also known as supervised learning. A new test point in feature space can then be classified according to some classifier decision rule. The available data (both normal and anomalous) should define the underlying distribution, or the classifier may be prone to overfitting.

- **Type III:** In many cases of anomaly detection, the ratio of anomalous to normal data will be small. As the a large amount of normal data should approximately describe the support of all possible normal points in feature space, it is easier to use an algorithm that defines this support. This known as semi-supervised learning, where we only train on normal data points. Any test samples falling inside the boundary will be classified as normal, whilst all other points will be classified as anomalies.

As we had a labelled dataset, we were going to either be creating a Type II or a Type III detection method. There are three general approaches to solving our problem.

1. In a naïve approach of creating a Type II detector, we could weight the classes and make our model preferential to the minority class. As a simple example, we could create a logistic regression model, and weight it towards the seizure class.

2. We could use undersampling, oversampling or a combination of both. There are many techniques already implemented in libraries like random undersampling, ADASYN, and SMOTE/ENN (Synthetic over sampling followed by Edited Nearest Neighbours used to pare down and centralize the anomalous data points). These techniques, along with manual synthetic data generation methods (rotation, translation, and dilation of feature vectors) are valid ideas

that have worked in other domains [36], but will not be the focus of this thesis. In this approach, the new dataset would then be used to create another Type II algorithm.

3. Instead of trying to balance the dataset, we can try and just predict the outlier class using anomaly detection techniques. There have been a variety of anomaly detection methods available over the years, and they have the advantage of not trying to sample from an underlying distribution of data to forcefully balance a dataset, and have shown very good results in applications like intrusion detection, system health monitoring, fraud detection and fault detection in complex operating environments. This would be a Type III algorithm.

In this thesis, we have decided to approach the problem of seizure detection from the perspective of anomaly detection due to the robustness it can provide, especially when we move on to detecting more subtle seizure subtypes like FUS. Additionally there have been decent results in literature [37] using similar sensor stacks (though detection generally occurs retrospectively).

Detection Method Motivation

We have explored 5 different anomaly detection methods in this thesis to apply to our problem.

SVM/OCSVM: Support Vector Machines have been widely used in classification tasks and can show surprisingly good performance. It has a high complexity (between $O(n^2)$ and $O(n^3)$) especially if you use kernel-SVM, but can still provide a good baseline for our other classifiers. Note as we are using SVM for anomaly detection, we will use a slight variation called One-Class SVM (OCSVM).

SVDD: Support Vector Data Description (SVDD) has a similar problem setup to SVM, but instead of attempting to generate a hyperplane, to separate points, it generates a hypersphere. This makes it well-suited to anomaly detection tasks, and it has been used in literature to good effect [38], [39].

Isolation Forest: As we eventually want a real time detection mechanism implemented on a portable device, the low complexity and memory requirement of the isolation forest stood out. It performs best on low sample sizes during training, has inherently low bias and variance, and any dependencies between features will not affect its performance.

Extended Isolation Forest: A natural extension to isolation forests, which can cause artefacts in certain areas of the search space, due to the orthogonal nature of how splits are made. The extended isolation forest takes advantage of the high dimensionality of the data by creating random hyperplane splits across the search-space rather than only splitting on certain feature values.

Anomaly Detection Theory

One Class Support Vector Machine (OCSVM)

A well-known technique in classification is Support Vector Machines (SVM) created by Vapnik [40]. As it is easily implemented and available, we use it as a baseline for future algorithms. Like most pattern recognition functions, SVM aims to find a pattern in the training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l) \in \mathbb{R}^D \times \{\pm 1\}$, where D is the dimensionality of the input space and define a decision function $f(\cdot)$ whereby it can correctly classify a new sample $f(\boldsymbol{x}) = y$ generated from the same underlying process as the training set. SVM's accomplish this task be creating an optimal hyper-plane between all training samples corresponding to the two classes $y = +1$ and $y = -1$.

First define the family of hyperplanes that SVM can model. Defining $\boldsymbol{w} \in \mathbb{R}^D$
as the weight vector normal to the hyperplane, and $\boldsymbol{x} \in \mathbb{R}^D$ a point in D-dimensional space, we see that $\boldsymbol{w} \cdot \boldsymbol{x} = 0$ will describe the locust of points defining the hyperplane passing through the origin that is orthogonal to $\boldsymbol{w}$. Extending this to the general case, we can define all hyperplanes as $\boldsymbol{w} \cdot \boldsymbol{x} = b$, or equivalently $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$, where $b \in \mathbb{R}$ is the bias term. This is the definition of the general hyperplane equation [41].

All point will then lie on either side of this hyperplane. $\boldsymbol{w} \cdot \boldsymbol{b} < 0$ will refer to points on the origin side of the hyperplane which we will classify as -1, whilst $\boldsymbol{w} \cdot \boldsymbol{x} + b > 0$ will refer to points on the opposite side, which we will classify as $+1$. The corresponding decision function is then $f(x) = sign((\boldsymbol{w} \cdot \boldsymbol{x}) + b)$.

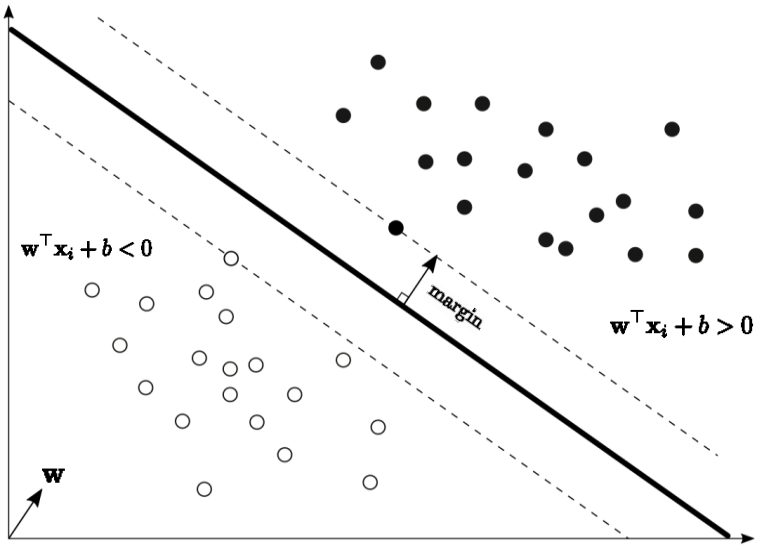


Figure 4: SVM margin and decision boundary

Linearly separable datasets will have an infinite number of hyperplanes that can perfectly separate them. To find the best separating hyperplane, we try to maximize the distance from the decision surface to the closest data point. This distance is also known as the margin.

Functional Margin: We can naively determine a margin with respect to a training example as $\hat{m}_i = y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b)$. This will always be positive (both y_i and $\boldsymbol{w} \cdot \boldsymbol{x} + b$ will both simultaneously be either positive or negative). From the definition of margin in the previous paragraph, the functional margin can be defined as the minimum margin

$$\hat{m} = \min_{i=1\ldots N} \hat{m}_i$$

where N is the number of data points. The issue with the functional margin is that $c\boldsymbol{w} \cdot \boldsymbol{x} = cb$ and $\boldsymbol{w} \cdot \boldsymbol{x} = b$ define the same hyperplane. It follows that we can set $\hat{m}_i = y_i(c\boldsymbol{w} \cdot \boldsymbol{x} + cb)$ to be as arbitrarily large as we want to without violating the problem formulation.

Geometric Margin: As c can be set to any arbitrary number, we set it to scale the functional margin of the points that determine $\hat{m}$. For those points lying closest to the decision plane, also known as the support vectors, $y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) = 1$. We are fixing c to make this true. Note that 1 is used as a reference only due to mathematical convenience and we could have used any other positive real number.

Our new decision rule for all points is

$$y_i = \begin{cases} -1 & if\ \boldsymbol{w} \cdot \boldsymbol{x_i} + b \leq -1 \\ +1 & if\ \boldsymbol{w} \cdot \boldsymbol{x_i} + b \geq +1 \end{cases}$$

$$\boldsymbol{OR}$$

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1$$

Note how this rule does not allow any points inside the margin. This means that the training data we provide must be linearly separable, otherwise this setup will fail. We will soon introduce a method to relax this constraint.

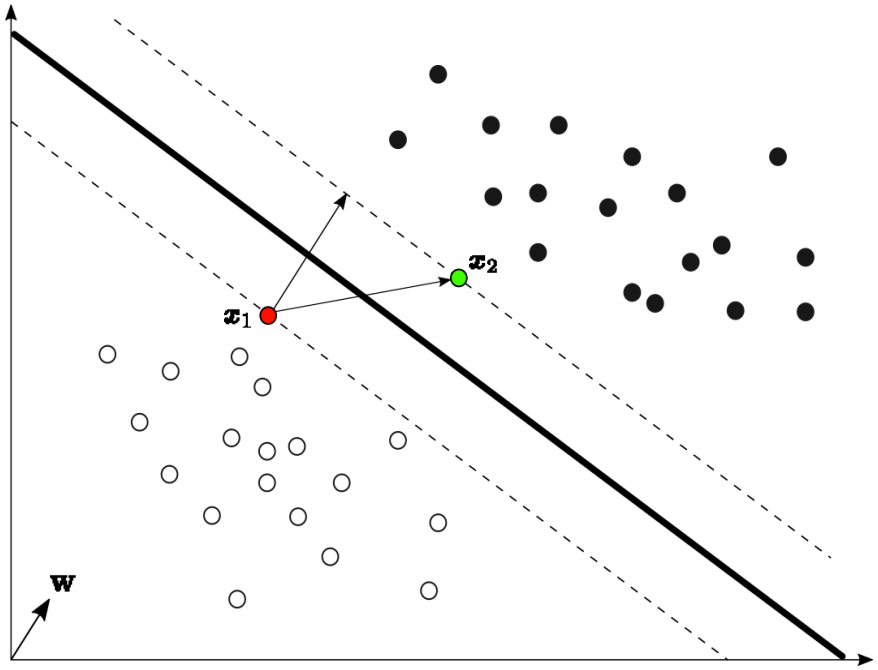


Figure 5: SVM decision boundary showing two support vectors on margins

We define x_1 and x_2 as two points on the margins. As the goal is to maximize the margin, the closest points to the hyperplane from both classes will be equidistant, so can assume $\{\boldsymbol{x_1}, -1\}$ and $\{\boldsymbol{x_2}, +1\}$.

$$\boldsymbol{w} \cdot \boldsymbol{x_1} + b = -1$$

$$\boldsymbol{w} \cdot \boldsymbol{x_2} + b = 1$$

$$\therefore \boldsymbol{w}(\boldsymbol{x_2} - \boldsymbol{x_1}) = 2$$

$\boldsymbol{x_2} - \boldsymbol{x_1}$ will traverse the margin, though it will not necessarily be orthogonal to the hyperplane. Since $\boldsymbol{w}$ is orthogonal to the hyperplane, the projection of $\boldsymbol{x_2} - \boldsymbol{x_1}$ onto $\boldsymbol{w}$ will give us the length of the margin.

$$\text{margin} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}(\boldsymbol{x_2} - \boldsymbol{x_1}) = \frac{2}{\|\boldsymbol{w}\|}$$

The optimization problem can now be setup as

$$\max_{\boldsymbol{w},b} \frac{2}{\|\boldsymbol{w}\|}$$

$$\text{s.t. } y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1 \geq 0$$

As numerically optimization packages are typically setup to minimize convex functions, we turn this maximization into a minimization problem.

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$s.t. \; y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1 \geq 0, \forall i \in \mathbb{N}$$

Note that as $\|\boldsymbol{w}\|$ will be a strictly positive number, and as squaring is a monotonic operator, this is valid. We square $\|\boldsymbol{w}\|$ to get rid of the $\frac{1}{2}$ coefficient when taking the derivative in the future.

We can incorporate the constraint into our minimization by setting up a Lagrangrian

$$\mathcal{L}(\boldsymbol{w}, b) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{\boldsymbol{i}} \alpha_i \left[(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1)\right]$$

$$\alpha_i \geq 0, \forall i \in \mathbb{N}$$

The primal optimization is then

$$p^* = \min_{\boldsymbol{w},b} \max_{\alpha_i \geq 0} \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{\boldsymbol{i}} \alpha_i \left[(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1)\right]$$

It is readily seen how the Lagrangian enforces our constraint in the primal form. Since we are searching for the α_i values that maximize the objective function, if the constraint is met, all of the $\alpha_i = 0$ in the case of non-support vectors and $(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1) = 0$ in the case of the support vectors. Optimization will once again reduce the objective to $\underset{\boldsymbol{w},b}{min} \frac{1}{2}\|\boldsymbol{w}\|^2$. If one or more of the constraints are violated, the corresponding α_i's of those points will tend to ∞, causing the objective to go to ∞. Since we are eventually minimizing with respect to $\boldsymbol{w}, b$, the solution will always contain appropriate values to enforce $(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1 \geq 0)$ to prevent this from happening.

We define the dual formulation of this problem as

$$d^* = \max_{\alpha_i \geq 0} \min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{\boldsymbol{i}} \alpha_i \left[(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1)\right]$$

We motivate the dual by first defining **weak duality**, which states that for any general problem (not necessarily convex), $p^* \geq d^*$. This is also known as the minimax inequality.

Proof: For any function ϕ of vector variables x, y

$$\textbf{Define } g(x) \triangleq \min_y \phi(x,y)$$

$$g(x) \leq \phi(x,y)$$

$$\max_x g(x) \leq \max_x \phi(x,y)$$

$$\max_x \min_y \phi(x,y) \leq \max_x \phi(x,y)$$

$$\max_x \min_y \phi(x,y) \leq \min_y \max_x \phi(x,y)$$

Note that as $\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{\boldsymbol{i}} \alpha_i \left[(y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)-1)\right]$ is a pointwise minimization of affine functions, it is concave [42]. Since we maximize over $\alpha_i's$ in the next step, the dual problem will always be a convex optimization [43].

We define $p^* - d^*$ as the duality gap, which will always be positive in the case of weak duality. We define a 0 duality gap as **strong duality**, meaning the solving the primal gives us the same answer as solving the dual. From Boyd [42], we know that strong duality holds if the chosen parameters meet the Karush-Kuhn-Tucker (KKT) conditions [42]–[44] identified below in the context of our problem

1. **Stationarity:** $\nabla_w \mathcal{L}(\boldsymbol{w}, b, \alpha) = 0$ and $\frac{\partial \mathcal{L}(\boldsymbol{w},b,\alpha)}{\partial b} = 0$
2. **Dual Feasibility**: $\alpha_i \geq 0$
3. **Primal Feasibility**: $(y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)-1) \geq 0$
4. **Complementary Slackness**: $\alpha_i(y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)-1) = 0$

From [44], we see that by solving the optimization such that it meets the KKT conditions, we can assume **strong duality**, defined as where optimizing both the primal and dual will give the same solution, $p^* = d^*$. We have already met conditions 2, 3 and 4 in the setup. Finally notice how if we take solve the inner optimization of the dual, we will satisfy condition 1.

$$\frac{\partial}{\partial \boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \alpha) = \boldsymbol{w} - \sum_i \alpha_i y_i \boldsymbol{x}_i = 0$$

$$\therefore \boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$$

$$\frac{\partial}{\partial b} \mathcal{L}(\boldsymbol{w}, b, \alpha) = 0 - \sum_i \alpha_i y_i = 0$$

$$\therefore \sum_i \alpha_i y_i = 0$$

We can then simplify the dual problem using these solutions

$$\frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{\boldsymbol{i}} \alpha_i \left[(y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)-1)\right]$$

$$= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} - \sum_{\boldsymbol{i}} \alpha_i\, y_i \boldsymbol{w}\cdot\boldsymbol{x_i} - b\sum_{\boldsymbol{i}} \alpha_i\, y_i + \sum_{\boldsymbol{i}} \alpha_i$$

$$= \frac{1}{2}\sum_{\boldsymbol{i,j}} \alpha_i\, \alpha_j y_i y_j \boldsymbol{x_i}\cdot\boldsymbol{x_j} - \sum_{\boldsymbol{i,j}} \alpha_i\, \alpha_j y_i y_j \boldsymbol{x_i}\cdot\boldsymbol{x_j} - b\sum_{\boldsymbol{i}} \alpha_i\, y_i + \sum_{\boldsymbol{i}} \alpha_i$$

$$= \sum_{\boldsymbol{i}} \alpha_i - \frac{1}{2}\sum_{\boldsymbol{i,j}} \alpha_i\, \alpha_j y_i y_j \boldsymbol{x_i}\cdot\boldsymbol{x_j}$$

Leading to our final dual form:

$$\max_{\alpha_i} \sum_{\boldsymbol{i}} \alpha_i - \frac{1}{2}\sum_{\boldsymbol{i,j}} \alpha_i\, \alpha_j y_i y_j \boldsymbol{x_i}\cdot\boldsymbol{x_j}$$

$$s.t. a_i \geq 0$$

$$\sum_i \alpha_i y_i = 0$$

This can now be solved using numerical optimization algorithms such as SMO [43]–[45], a commonly used algorithm to solve quadratic programming problems. Proof of SMO is out of the scope of this thesis, though it is implemented by multiple Quadratic Programming Solvers and SVM libraries. Note that the program will return us optimal for α_i, most of which will be 0 (only support vectors will have non-zero $\alpha_i's$). We can use these to calculate $\boldsymbol{w}$, and then use the complementary slackness condition to calculate

b. Then for any new test point, we can classify it according to $y_{test} = sign(\boldsymbol{w}^\top \boldsymbol{x} + b)$. By convention, any point that falls on the hyperplane boundary will generally go to the positive class [46].

Soft-Margin SVM (Non-separable case)

With the current derivation of SVM, the optimization will fail if the data is non-separable. Additionally, any separable data with outliers will dramatically shift the decision hyperplane. In this case we can add in some slack parameters to relax the strict separability condition, leading our optimization problem to become

$$\min_{\xi,\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{\boldsymbol{i}} \xi_i$$

$$s.t.\ \ y_i(\boldsymbol{w}\cdot \boldsymbol{x_i} + b) - 1 \geq \xi_i$$

$$\xi_i \geq 0, \forall i \in \mathbb{N}$$

The C parameter controls relative weighting on the penalizing term, and can be set during tuning. High C values will force the SVM to strictly enforce the margins, whilst small values will allow for more miss-classification. Also note the constraint on ξ_i. Only points that break the margin will have a value for ξ_i, with points on the correct side of the margin having $\xi_i = 0$. This is known as hinge loss and it is non-differentiable. Primal solutions can make use of the sub-gradient, whilst dual solutions will result in a quadratic problem. This formulation is known as L1-SVM, and enforces sparsity in the solution. L1-SVM can be used in deep learning models using sub gradient descent, or we could make the hinge loss differentiable by using an L2-SVM with an objective of $\min_{\xi,\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{\boldsymbol{i}} \xi_i^2$ [47].

To solve the L1-SVM formulation, we follow the same process as the original SVM. Setting up the Lagrangian,

$$\mathcal{L}(\boldsymbol{w}, b, \xi, \alpha, r) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{\boldsymbol{i}} \xi_i - \sum_{\boldsymbol{i}} \alpha_i \left[(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) + \xi_i - 1)\right] - \sum_{\boldsymbol{i}} r_i \xi_i$$

Optimizing the Lagrangian w.r.t $\boldsymbol{w}, b, \xi$,

$$\frac{\partial}{\partial \boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \xi, \alpha, r) = \boldsymbol{w} - \sum_{i} \alpha_i y_i \boldsymbol{x_i} = 0$$

$$\therefore \boldsymbol{w} = \sum_{i} \alpha_i y_i \boldsymbol{x}_i$$

$$\frac{\partial}{\partial b} \mathcal{L}(\boldsymbol{w}, b, \alpha) = 0 - \sum_{i} \alpha_i y_i = 0$$

$$\therefore \sum_{i} \alpha_i y_i = 0$$

$$\frac{\partial}{\partial \boldsymbol{\xi}} \mathcal{L}(\boldsymbol{w}, b, \xi, \alpha, r) = \ C - \alpha_i - r_i = 0$$

$$\therefore C = \alpha_i + r_i$$

Substituting and rearranging the Lagrangian

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{\boldsymbol{i}} \xi_i - \sum_{\boldsymbol{i}} \alpha_i \left[(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) + \xi_i - 1)\right] - \sum_{\boldsymbol{i}} r_i \xi_i$$

$$= \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{\boldsymbol{i}} \alpha_i \xi_i + \sum_{\boldsymbol{i}} r_i \xi_i - \sum_{\boldsymbol{i}} \alpha_i \left[(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1)\right] - \sum_{\boldsymbol{i}} \alpha_i \xi_i - \sum_{\boldsymbol{i}} r_i \xi_i$$

$$= \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} - \sum_{\boldsymbol{i}} \alpha_i \left[(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1)\right]$$

$$= \frac{1}{2}\sum_{\boldsymbol{i,j}} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j} - \sum_{\boldsymbol{i,j}} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j} - b\sum_{\boldsymbol{i}} \alpha_i y_i + \sum_{\boldsymbol{i}} \alpha_i$$

$$= \sum_{\boldsymbol{i}} \alpha_i - \frac{1}{2} \sum_{\boldsymbol{i,j}} \alpha_i \, \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}$$

Note that this is exactly the same Lagrangian that we got in the hard-margin SVM. The difference will be in the constraints, as we now have $C = \alpha_i + r_i$. With the knowledge that $\alpha_i \geq 0$ and $r_i \geq 0$ (from the dual feasibility KKT condition), we can deduce that $0 \leq \alpha_i \leq C$, as $C - r_i = \alpha_i$ must be true. This leads to the final dual setup

$$\max_{\alpha} \sum_{\boldsymbol{i}} \alpha_i - \frac{1}{2} \sum_{\boldsymbol{i,j}} \alpha_i \, \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}$$

$$s.t. \;\; 0 \leq \alpha_i \leq C, \forall i \in \mathbb{N}$$

$$\sum_i \alpha_i y_i = 0$$

This dual problem can once again be solved using a quadratic program, but this time will not break if the dataset is not linearly separable. As before, once the boundary is made, to test a new point, we use $y_{test} = sign(\boldsymbol{w}^\top \boldsymbol{x} + b)$, with any point on the boundary going to the +1 class.

Kernel Methods

Due to how SVMs are setup, they have the ability to find non-linear hyperplanes by projecting the data and performing the optimization in a higher dimensional space. This is known as the kernel trick. To investigate this, let's say we have a 1-dimensional dataset, where each value $x \in \mathbb{R}$. Let's also assume that the dataset is non-linearly separable, but if we transform all the points to the 2D space with $\phi(x) = (x, x^2)$, it is linearly separable. We can simply project all of the points to this 2D space, and perform the entire optimization there, essentially solving

$$\max_{\alpha} \sum_{\boldsymbol{i}} \alpha_i - \frac{1}{2} \sum_{\boldsymbol{i,j}} \alpha_i \, \alpha_j y_i y_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j})$$

$$s.t. \;\; 0 \leq \alpha_i \leq C, \forall i \in \mathbb{N}$$

$$\sum_i \alpha_i y_i = 0$$

After solving the Quadratic Program, we will use

$$y^{test} = \text{sign}(\boldsymbol{w} \cdot \phi(\boldsymbol{x}^{test}) + b)$$

$$= \text{sign}(\sum_i \alpha_i y_i \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_{test}) + b)$$

as a decision rule, where we were able to substitute $\boldsymbol{w} = \sum_i \alpha_i y_i \phi(\boldsymbol{x}_i)$ to form an inner product in the transformed space. Projecting every variable to a higher dimensional space and then computing the dot product is inefficient. There exists a family of functions called **kernels** that can compute the inner product in a higher dimensional feature space at a low cost. As all of the $\boldsymbol{x}$ values involved in optimization and testing are used in inner products, this is an efficient way of computing non-linear decision boundaries for your dataset.

For a kernel to be valid, it has to obey **Mercer's condition**, which states that for any kernel, it is necessary and sufficient that for $\{x_1, \ldots, x_m\}, (m < \infty)$, the corresponding kernel matrix is symmetric and positive semi-definite [43], [44], [48]. You can build your own kernel and then test these conditions retrospectively to test for validity, but this is often tricky due to the difficulty of testing for positive semi-definiteness.

Another option is to create kernels by construction. As a simple example, take two vectors $\boldsymbol{x} = (x_1, x_2)$, and $\boldsymbol{y} = (y_1, y_2)$, giving the inner product $\boldsymbol{x}^\top \boldsymbol{y} = (x_1 y_1, x_2 y_2)$. If we square this dot product, we will get pairwise multiplications of each term $(\boldsymbol{x}^\top \boldsymbol{y})^2 = x_1^2 y_1^2 + x_1 x_2 y_1 y_2 + x_1 x_2 y_1 y_2 + x_2^2 y_2^2$. This is the same as projecting the vectors to a a quadratic feature space, $\phi(\boldsymbol{x}) = (x_1^2,\ x_1 x_2,\ x_2 x_1,\ x_2^2)$, $\phi(\boldsymbol{y}) = (y_1^2,\ y_1 y_2,\ y_2 y_1,\ y_2^2)$. Note how $(\boldsymbol{x}^\top \boldsymbol{y})^2 = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$. We call $K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y})^2$ a kernel function. It calculates the value of the dot product in the quadratic space without having to actually transform $\boldsymbol{x}$ and $\boldsymbol{y}$ to the quadratic space.

This may seem like a vacuous step, but we can extend the notion to n-dimensional inputs with m-degree polynomials. If we redefine the input vectors to be $\boldsymbol{x} = (x_1, \dots, x_n)$, and $\boldsymbol{y} = (y_1, \dots, y_n)$, we can describe a generalized m'th degree polynomial kernel to be

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y})^m = \left(\sum_i x_i y_i \right)^m = \sum_{i_1, i_2, \dots, i_m} x_{i_1} x_{i_2} \dots x_{i_m} y_{i_1} y_{i_2} \dots y_{i_m}$$

This is the dot product in the m'th polynomial space, and is relatively easy to calculate, whilst calculating the actual vector projections in that space become extremely complex. The kernel mentioned above is known as the polynomial kernel with no offset. The most popular kernels to start off with are

- **Polynomial**: $K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y} + b)^m$
- **Sigmoid**: $K(\boldsymbol{x}, \boldsymbol{y}) = (\alpha \boldsymbol{x}^\top \boldsymbol{y} + c)$
- **Radial Basis Function (Gaussian)**: $K(\boldsymbol{x}, \boldsymbol{y}) = exp(-\frac{\|x-y\|^2}{2\sigma^2})$

Note there are many more valid kernels available. The Gaussian RBF kernel is especially popular as a baseline due to its characteristics of stationarity (or translation invariance, $K(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{x} + \boldsymbol{c}, \boldsymbol{y} + \boldsymbol{c})$),

and isotropicity (scaling by the σ parameter occurs by the same amount in all directions). They also work well in practice and are very easy to tune (only 1 parameter in the search space), compared to other kernels.

Nu-SVM

There is another possible realization of a soft-margin SVM known as ν-SVM [49], [50]. Instead of C, we use a new parameter $\nu \in (0,1]$ that will set an upper and lower bound on the number of support vectors on the wrong side of the hyperplane. As it is similar to C-SVM, we start with the primal objective

$$\min_{\boldsymbol{w},b,\xi,\rho} \frac{1}{2}\|\boldsymbol{w}\|^2 - \nu\rho + \frac{1}{N}\sum_i \xi_i$$

$$s.t.\ \ y_i(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x_i}) + b) \geq \rho - \xi_i, \forall i \in \mathbb{N}$$

$$\xi_i \geq 0, \rho \geq 0$$

We can see from the first constraint that the parameter ρ sets the margin of the SVM, and in this case that margin will be $\frac{2\rho}{\|\boldsymbol{w}\|}$. Only points inside this margin will be penalized according to the constraint. This was not possible in the C-SVM, where points were penalized as soon as they broke the set geometric margin of 1. Additionally note how by tuning the parameter ν we can encourage ρ to grow faster. The increase in ρ will be counteracted by a commensurate increase in ξ_i which will be penalized. This is how the tension in this optimization is setup.

It has several important properties, proved in [49], but they are not going to be covered in this thesis as both types of soft margin SVM have similar classification powers, and the justification of the primal from first principles is more involved. Once the primal has been found as shown above, we follow the same steps as with C-SVM to obtain the following dual [50].

$$\min_{\alpha} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j})$$

$$s.t. \;\; 0 \leq \alpha_i \leq \frac{1}{N}, \forall i$$

$$\sum_i \alpha_i y_i = 0$$

$$\sum_i \alpha_i \geq \nu$$

with the decision function $y^{test} = sign(\sum_i \alpha_i y_i \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_{test}) + b)$. As before, it's the same objective and evaluation function, with some adjustments to the constraints.

The main advantage of ν-SVM is that the ν parameter to define a specific number of support vectors that you may want. This can help in data classification tasks. Additionally, ν-SVM has also been implemented in many common libraries. In practice, ν-SVM has been known to be slightly trickier to tune in comparison with C-SVM, and both show similar results if tuned properly.

One-Class SVM

In cases of anomaly detection, using SVMs as we have derived them in the previous sections can lead to inaccurate decision boundaries, as the underlying distribution of the anomaly data will not be well represented in our training set. Scholkopf et. al. [46] addressed this issue by slightly modifying the original SVM formulation. The strategy was to map the feature values to some kernel space, and then find a separating hyperplane that separates them from the origin with maximum margin. This will create a decision function that will capture most of the data points within a small region in input space, labelling any test point that falls into that region as +1. Any point that falls outside of that region will be set as -1.

Due to the large sampling of normal data, we can assume that it represents the support reasonably well, and as such anything outside will be classified as an anomaly. We will walk through the setup of this type of SVM, but due to its similarities with the previous sections, certain steps will be omitted.

We begin with the primal. Since we want to maximize the margin from the origin, we decide to exclude the bias term b as then the hyperplane is guaranteed to pass through the origin. Like in the ν-SVM setup, we set a free parameter ρ for the margin, and $\nu \in (0,1]$ will be a hyperparameter that controls the penalization term for points that break the margin. A $\nu \to 1$ will allow for a lot of slack, whilst $\nu \to 0$ will correspond to a hard margin setup. Also note the lack of y_i term, as we are only training with positive samples.

$$\min_{\boldsymbol{w},\boldsymbol{\xi},\rho} \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{\nu N}\sum_i \xi_i - \rho$$

$$s.t. \quad \boldsymbol{w}\cdot\phi(\boldsymbol{x_i}) \geq \rho - \xi_i,$$

$$\xi_i \geq 0, \forall i \in \mathbb{N}$$

With this decision boundary, we will expect that most of the training data will be mapped to +1 with the decision function $sign(\boldsymbol{w}\cdot\phi(\boldsymbol{x}) - \rho)$. The Lagrangian can be setup as

$$\mathcal{L}(\boldsymbol{w},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta},\rho) = \min_{\boldsymbol{w},\boldsymbol{\xi},\rho} \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{\nu N}\sum_i \xi_i - \rho - \sum_i \alpha_i(\boldsymbol{w}\cdot\phi(\boldsymbol{x_i}) - \rho + \xi_i) - \sum_i \beta_i\xi_i$$

With the derivatives

$$\frac{\partial}{\partial \boldsymbol{w}}\mathcal{L}(\boldsymbol{w},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta},\rho) = \boldsymbol{w} - \sum_i \alpha_i\phi(\boldsymbol{x_i}) = 0$$

$$\therefore \boldsymbol{w} = \sum_i \alpha_i \phi(\boldsymbol{x}_i)$$

$$\frac{\partial}{\partial \boldsymbol{\xi}} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \rho) = \frac{1}{\nu N} - \alpha_i - \beta_i = 0$$

$$\therefore \alpha_i = \frac{1}{\nu N} - \beta_i$$

$$\frac{\partial}{\partial \rho} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \rho) = -1 + \sum_i \alpha_i$$

$$\therefore \sum_i \alpha_i = 1$$

From the KKT conditions, we know $\alpha_i, \beta_i \geq 0$, and using this we can simplify the middle inequality to $0 \leq \alpha_i \leq \frac{1}{\nu N}$. Substituting these values back into the objective, we can simplify it

$$\frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} + \frac{1}{\nu N}\sum_i \xi_i - \rho - \sum_i \alpha_i (\boldsymbol{w} \cdot \phi(\boldsymbol{x_i}) - \rho + \xi_i) - \sum_i \beta_i \xi_i$$

$$= \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j}) + \frac{1}{\nu N}\sum_i \xi_i - \sum_{i,j} \alpha_i \alpha_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j}) - \sum_i \alpha_i \xi_i - \sum_i \beta_i \xi_i$$

$$= -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j}) + \sum_i \frac{1}{\nu N} \xi_i - \sum_i (\alpha_i + \beta_i)\xi_i$$

$$= -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j})$$

With this objective, the dual can now be written as

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j})$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq \frac{1}{\nu N}$$

$$\sum_i \alpha_i = 1$$

with the decision function $\text{sign}(\boldsymbol{w} \cdot \phi(\boldsymbol{x}) - \rho)$

From [43], and the complementary slackness KKT conditions, we know that the support vectors will only exist when $0 < \alpha_i < \frac{1}{\nu N}$ (Not the strict inequalities). We also know that on the support vectors, $\xi_i = 0$. So to find ρ, we find our support vectors from our quadratic program, then use $\rho = \left(\boldsymbol{w} \cdot \phi(\boldsymbol{x_i})\right) = \sum_j \alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j})$.

Finally, notice how like all SVM methods, we can use the kernel trick to project the data into higher dimensional space. For one-class SVM RBF is generally a good choice for a kernel as it will give an enclosed boundary to define support of your data.

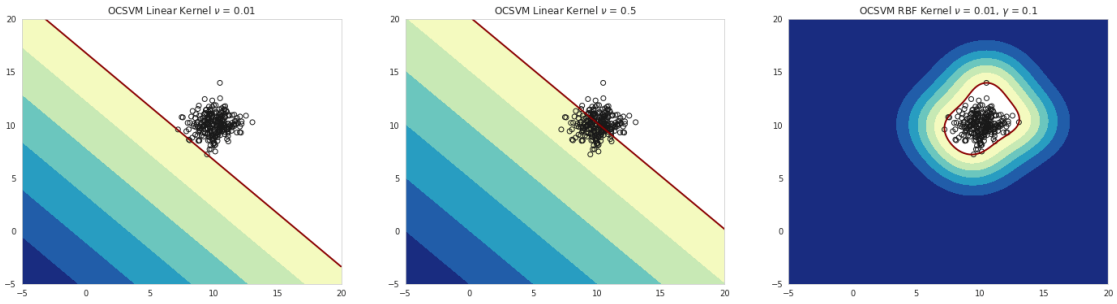


Figure 6: OCSVM implementation examples. Colored bands show distance of points from the boundary, with white being anything inside the boundary. a) shows OCSVM with a linear kernel on a 2D dataset. Notice how the boundary separates the dataset from the origin. b) shows same kernel with a higher slack, allowing more points to break the boundary. c) shows same dataset with an RBF kernel

Support Vector Data Description (SVDD)

As in one-class SVM, support vector data description aims to be targeted towards anomaly detection by describing the support of a distribution of the given data [51]. The strategy of SVDD is not to attempt to model the perfect support of the target dataset. Rather, we model a spherically shaped boundary around the target set, and minimize the radius of this sphere to maximize the possibility of outlier detection.

We approach the problem in a similar manner to other support vector problems. First define a training set of only "normal" data datapoints $\{\boldsymbol{x_1}, \dots, \boldsymbol{x_n}\}, n \in \mathbb{N}$ for which we want to find the support. We then also assume that the data has some (ideally equal) variance in all of the feature dimensions of the input. This is an important assumption as if we model on a thin dataset, there will be a lot of room inside the sphere for anomalies to be captured. We then define a hypersphere with center $\boldsymbol{c}$ and radius $R > 0$, with the goal of SVDD being to minimize this sphere whilst demanding it contain all training samples.

As in the soft margin SVM, we will add some slack to the constraints to ensure that our optimization is possible. This gives the primal

$$\min_R R^2 + C\sum_i \xi_i$$

$$s.t. \quad \|\boldsymbol{x_i} - \boldsymbol{c}\|^2 \leq R^2 + \xi_i,$$

$$\xi_i \geq 0, \forall i \in \mathbb{N}$$

Note once more how the slack parameter creates the optimization problem we are searching for. It relaxes the constraint of every datapoint being inside the hypersphere, however the further away a datapoint is, the more it is penalized.

The corresponding Lagrangian formulation is

$$\mathcal{L}(R, \boldsymbol{a}, \alpha_i, \gamma_i, \xi_i) = R^2 + C\sum_i \xi_i - \sum_i \alpha_i[(\|\boldsymbol{x_i} - \boldsymbol{c}\|^2) + R^2 + \xi_i] - \sum_i \gamma_i \xi_i$$

$$= R^2 + C\sum_i \xi_i - \sum_i \alpha_i[(\boldsymbol{x_i} \cdot \boldsymbol{x_i} - 2\boldsymbol{c} \cdot \boldsymbol{x_i} + \boldsymbol{c} \cdot \boldsymbol{c}) + R^2 + \xi_i] - \sum_i \gamma_i \xi_i$$

Where according to the KKT conditions, $\alpha_i \geq 0, \gamma_i \geq 0$. In the dual, we minimize the Lagrangian w.r.t. $R, \boldsymbol{c}$ and ξ_i.

$$\frac{\partial \mathcal{L}(R, \boldsymbol{a}, \alpha_i, \gamma_i, \xi_i)}{\partial R} = 2R - 2R\left(\sum_i \alpha_i\right) = 0$$

$$\therefore \sum_i \alpha_i = 1$$

$$\frac{\partial \mathcal{L}(R, \boldsymbol{a}, \alpha_i, \gamma_i, \xi_i)}{\partial \boldsymbol{c}} = 2\sum_i \alpha_i x_i - 2\boldsymbol{c}\sum_i \alpha_i = 0$$

$$\therefore \boldsymbol{c} = \frac{\sum_i \alpha_i \boldsymbol{x_i}}{\sum_i \alpha_i} = \sum_i \alpha_i \boldsymbol{x_i}$$

$$\frac{\partial \mathcal{L}(R, \boldsymbol{a}, \alpha_i, \gamma_i, \xi_i)}{\partial \xi_i} = C - \alpha_i - \gamma_i = 0$$

Note that in the last partial, we take the derivative w.r.t. a single ξ_i. This is why the summation does not matter, all other terms in the sum except for this single ξ_i will be constant. Substituting these formulations back into our original Lagrangian

$$\mathcal{L}(R, \boldsymbol{a}, \alpha_i, \gamma_i, \xi_i) = R^2 + C\sum_i \xi_i - \sum_i \alpha_i[(\boldsymbol{x_i} \cdot \boldsymbol{x_i} - 2\boldsymbol{c} \cdot \boldsymbol{x_i} + \boldsymbol{c} \cdot \boldsymbol{c}) + R^2 + \xi_i] - \sum_i \gamma_i \xi_i$$

$$= R^2 + \sum_i \alpha_i \xi_i + \sum_i \gamma_i \xi_i - \sum_i \alpha_i(\boldsymbol{x_i} \cdot \boldsymbol{x_i} - 2\boldsymbol{c} \cdot \boldsymbol{x_i} + \boldsymbol{c} \cdot \boldsymbol{c}) - R^2 - \sum_i \alpha_i \xi_i - \sum_i \gamma_i \xi_i$$

$$= -\sum_i \alpha_i(\boldsymbol{x_i} \cdot \boldsymbol{x_i}) + 2\sum_i \alpha_i(\boldsymbol{c} \cdot \boldsymbol{x_i}) - \boldsymbol{c} \cdot \boldsymbol{c}$$

$$= \sum_i \alpha_i(\boldsymbol{x_i} \cdot \boldsymbol{x_i}) - \sum_{i,j} \alpha_i \alpha_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

Since we know that $\alpha_i = C - \gamma_i, \alpha_i \geq 0, \gamma_i \geq 0$, we can incorporate all this information into a single inequality, $0 \leq \alpha_i \leq C$. The dual quadratic program can thus be written as

$$\min_{\boldsymbol{\alpha}} \sum_{i,j} \alpha_i \alpha_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j}) - \sum_i \alpha_i (\boldsymbol{x_i} \cdot \boldsymbol{x_i})$$

$$s.t. \quad 0 \leq \alpha_i \leq C$$

$$\sum_i \alpha_i = 1$$

As in soft-margin SVM, from the complementary slackness KKT conditions, we know that the Lagrange multiplier will be 0 when the data point satisfies the constraint, and maximized when the constraint is broken. Any support vector will have a value that is in between 0 and the maximum. As α_i is bounded by C, we know that all support vectors will have a $0 < \alpha_i < C$. Note the strict inequality in this case.

We can calculate $\boldsymbol{c}$ from the derived equation when setting the partial derivatives. We can calculate R^2 by computing the distance from a support vector to $\boldsymbol{c}$

$$R^2 = \|\boldsymbol{x_{SV}} - \boldsymbol{c}\|^2$$

$$= \boldsymbol{x_{SV}} \cdot \boldsymbol{x_{SV}} - 2\boldsymbol{c} \cdot \boldsymbol{x_{sv}} + \boldsymbol{c} \cdot \boldsymbol{c}$$

$$= \boldsymbol{x_{SV}} \cdot \boldsymbol{x_{SV}} - 2\sum_i \alpha_i(\boldsymbol{x_i} \cdot \boldsymbol{x_{SV}}) + \sum_{i,j} \alpha_i \alpha_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

At evaluation time, we compute the distance of a new point to the $\boldsymbol{c}$, and classify it accordingly. If it is inside the hypersphere then it is not an anomaly, otherwise it is. The decision function is

$$y_{test} = \text{sign}\left(-(\|\boldsymbol{x_{test}} - \boldsymbol{c}\|^2 - R^2)\right)$$
$$= \text{sign}\left(-\left(\boldsymbol{x_{SV}} \cdot \boldsymbol{x_{SV}} - 2\sum_i \alpha_i(\boldsymbol{x_i} \cdot \boldsymbol{x_{SV}}) + \sum_{i,j} \alpha_i \alpha_j(\boldsymbol{x_i} \cdot \boldsymbol{x_j}) - R^2\right)\right)$$

Note again how all of the $\boldsymbol{x}$ vectors appear as dot products, meaning we can also use the kernel trick with SVDD [51]. The ν version of SVDD can also be derived in a similar manner. We do not go through all the steps as they are identical to C-SVDD. Starting with the primal

$$\min_R R^2 + \frac{1}{\nu N}\sum_i \xi_i$$
$$s.t. \quad \|\boldsymbol{x_i} - \boldsymbol{c}\|^2 \leq R^2 + \xi_i,$$
$$\xi_i \geq 0, \forall i \in \mathbb{N}$$

After setting up the Lagrangian, taking the partials, substituting, and defining the new constraints, we are left with the quadratic dual equation [46]

$$\min_{\boldsymbol{\alpha}} \sum_{i,j} \alpha_i \alpha_j(\boldsymbol{x_i} \cdot \boldsymbol{x_j}) - \sum_i \alpha_i(\boldsymbol{x_i} \cdot \boldsymbol{x_i})$$
$$\text{s.t.} \quad 0 \leq \alpha_i \leq \frac{1}{\nu N}$$
$$\sum_i \alpha_i = 1$$

The same decision function as before can be used for testing new data points.

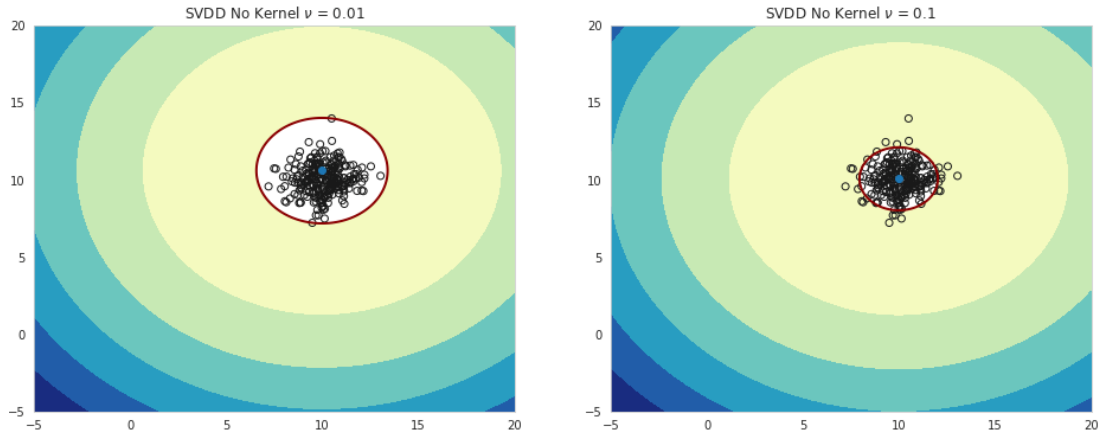


Figure 7: SVDD implementation example. OCSVM implementation examples. Colored bands show distance of points from the boundary, with white being anything inside the boundary. a) shows a vanilla SVDD with low slack. B) shows vanilla SVDD with higher slack, allowing more points to break the boundary.

SVM Implementations

SVM

When implementing SVM using quadratic programming solvers, it is often necessary to rearrange the data into a standard form that the solver will understand. In this thesis we used the quadratic programming solver **cvxopt**, which uses an interior-point method to solve the QP [52], [53]. Cvxopt expects the program to come in the form

$$\min_x \frac{1}{2} x^\top P x + q^\top x$$

$$\text{s.t. } Gx \leq h$$

$$Ax = b$$

To conform, we vectorize our objective function. For the soft-margin SVM, it would be

$$\max_{\alpha} \sum_{i} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j y_i y_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j})$$

$$= \min_{\alpha} \frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j y_i y_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j}) - \sum_{i} \alpha_i$$

$$= \min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i,j} \boldsymbol{\alpha}^{\top} \boldsymbol{P} \boldsymbol{\alpha} + (-\mathbb{I})^{\top} \boldsymbol{\alpha}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^{N \times 1}, \boldsymbol{P} \in \mathbb{R}^{N \times N}, \mathbb{I} \in \mathbb{R}^{N \times 1}$.

$$\boldsymbol{x}^{\top} = \boldsymbol{\alpha}^{\top} = [\alpha_1, \dots, \alpha_N]$$

$$\boldsymbol{q}^{\top} = \mathbb{I}^{\top} = [1, 1, \dots, 1]$$

$$\boldsymbol{P} = \begin{bmatrix} y_1 y_1 K(x_1, x_1) & \dots & y_1 y_N K(x_1, x_N) \\ \vdots & \ddots & \vdots \\ y_N y_1 K(x_N, x_1) & \dots & y_N y_N K(x_N, x_N) \end{bmatrix}$$

Note that to calculate the kernel matrix is tricky as we need to calculate half of the kernel values individually (All kernels are symmetric from Mercer's condition). This is one of the limitations of SVM as it will grow in a polynomial complexity.

The constraints are more challenging as we have an upper and lower bound on the inequality. To meet the constraint we create a large matrix $\boldsymbol{G}$ that matches the inequalities $-\alpha_i \leq 0, \alpha_i \leq C, \forall i$. The equality constraint can just be vectorized. Here $\boldsymbol{G} \in \mathbb{R}^{2N \times N}$, $\boldsymbol{h} \in \mathbb{R}^{2N \times 1}$

$$\boldsymbol{G} = \begin{bmatrix} -1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & -1 \\ 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

$$\boldsymbol{h}^{\top} = [0, \dots, 0, C, \dots, C]$$

In this way the equation $\boldsymbol{G\alpha} \leq \boldsymbol{h}$ will meet every inequality necessary. For the final inequality, we set up the vector of labels $\boldsymbol{A} \in \mathbb{R}^{1\times N}$ corresponding to ± 1, and $b = 0$ so that.

$$\boldsymbol{A\alpha} = b$$

OC-SVM

For OC-SVM we follow the same procedure to end up with the matrices

$$\boldsymbol{P} = \begin{bmatrix} K(x_1, x_1) & \dots & K(x_1, x_N) \\ \vdots & \ddots & \vdots \\ K(x_N, x_1) & \dots & K(x_N, x_N) \end{bmatrix}$$

$$\boldsymbol{q}^\top = [0, 0, \dots, 0]$$

$$\boldsymbol{G} = \begin{bmatrix} -1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & -1 \\ 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

$$\boldsymbol{h}^\top = \left[0, \dots, 0, \frac{1}{\nu N}, \dots, \frac{1}{\nu N}\right]$$

$$\boldsymbol{A} = [1, 1, \dots, 1]$$

$$b = 1$$

SVDD

$$\boldsymbol{P} = 2\begin{bmatrix} K(x_1, x_1) & \dots & K(x_1, x_N) \\ \vdots & \ddots & \vdots \\ K(x_N, x_1) & \dots & K(x_N, x_N) \end{bmatrix}$$

$$\boldsymbol{q} = -\mathrm{diag}(\boldsymbol{X} \cdot \boldsymbol{X}^\top)$$

$$\boldsymbol{G} = \begin{bmatrix} -1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & -1 \\ 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

$$\boldsymbol{h}^{\top} = \left[0, \ldots, 0, \frac{1}{\nu N}, \ldots, \frac{1}{\nu N}\right]$$

$$\boldsymbol{A} = [1, 1, \ldots, 1]$$

$$b = 1$$

Limitations of Support Vector Method

The biggest limitation with SVM lie in regard to the choice of kernel. Once the kernel is chosen, the user can only tune the error parameter, with the kernel hiding a lot of potentially critical information. Research is currently ongoing on choosing the ideal kernel for any particular problem, though the RBF kernel has been thought to be a good starting point for most datasets as it is stationary, isotropic and smooth.

SVM's can also sensitive to overfitting given specific kernels [54]. RBF's are especially notorious for this, as given a large enough γ parameter (or small σ), it can individually capture every positive point in the training set, leading to terrible generalization. The decisions are also "hard", in that a point is either an anomaly or it isn't. Many other deep learning and statistical methods can give likelihoods of class memberships for finer control.

Finally, complexity is a concern for SVM, in both training and testing. The complexity will depend on the type of SVM and kernel, though typical kernel SVM's will have a complexity between $O(n^2)$ and $O(n^3)$ for training [55], and $O(n_{SV}d)$ for runtime, where n_{SV} is the number of support vectors and d is the dimensionality (number of features) [56]. SVM's are limited by the size of the dataset as storing the kernel matrix will scale quadratically with the number of data points. The traditional algorithm is

infeasible in that scenario, however there have been some approximation methods (Nystrom approximation [57], Random Kitchen Sinks [58], and subsampling [59])

Isolation Forest

Decision trees are a popular machine learning algorithm due to their feature value scaling and transformation invariance, robustness against feature dependencies, and model interpretability. They are as close as we can get to an off-the-shelf data mining algorithm.

There are a number of conventional greedy methods generally used to grow decision trees, like ID3 (Iterative Dichotomiser 3), C4.5 and CART. Each has been built upon the foundations of the former, and CART is currently the most commonly implemented algorithm. All trees are built by starting off with a dataset of features and a classification or regression variable C. By iterating over each feature in the feature set and calculating a measure of uncertainty as to correctly predicting the C, the algorithm can decide upon the best feature to split that dataset. This uncertainty calculation and splitting is then recursively performed until we can separate C completely.

ID3 uses entropy $-\sum_{i=1}^{N} P(c_i)\, log\, P(c_i)$, and information gain (KL-Divergence) as a splitting criterion, and can only be used on nominal data. The splits do not have to be binary, i.e. a selected feature f_i with nominal values {Sunny, Rainy, Windy} can be split 3 ways. C4.5 performs splits on the greatest gain ratio, and can be performed on both nominal and continuous data. CART performs splits based on the Gini diversity index ($1 - \sum_i P_i^2$), and the decision CART decision trees are always binary. A few other differences between CART and C4.5 include pruning methods (simplifying decision trees to prevent overfitting, CART uses cost-complexity), and how each algorithm handles datasets with corrupted values, but they are out of the scope of this thesis [60].

In general, decision trees that are grown to have a large number of levels will have a overfit to a complicated decision boundary, leading to low bias but high variance. In decision, trees this can be mitigated either by pruning or bootstrap aggregation (bagging).

Consider a dataset $Z = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, where the x_i's are inputs, and the y_i's are outputs. We can fit a model $\hat{f}(\cdot)$ to this dataset, and obtain a prediction $\hat{f}(x)$ for a particular x. This prediction will have high variance, but bagging will average this prediction over a collection of bootstrap (subsample with replacement) samples from the dataset. This makes use of both the inherent randomness of the dataset, and the low bias of the estimator. For each bootstrap subsample Z^{*b}, $b = 1,2, \dots, B$ we fit an estimator $\hat{f}^{*b}(x)$, and then calculate the bagging estimate

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

Note that this is just the Monte Carlo approximation of the expectation of the estimator $E_{\hat{S}}[\hat{f}^*(x)]$, where $\hat{f}^*(\cdot)$ is an estimator fit to $Z^* = \{(x_1^*, y_1^*), (x_2^*, y_2^*), \dots, (x_N^*, y_N^*)\}$, with each $(x_i^*, y_i^*) \sim \hat{S}$. $\hat{S}$ is the uniform distribution of the tuples (x_i, y_i) from the original dataset. In other words, bagging finds the average estimator, and the low bias of the individual estimators ensures that as $B \to \infty$, we will get a well generalized (low-variance) model. This explanation was for regression random forests. In classification forests, you can use the same idea to create a voting system, where the class with the majority of estimators should be chosen.

Proof of Variance Decrease: In the case of the estimators being independent, let each estimator be represented by random variable X_i, with variance σ^2. We know the value of the average estimator is $\frac{1}{B}\sum_{i=1}^{B} X_i$, and want to find the variance of this average estimator.

$$\text{Var}\left(\frac{1}{B}\sum_{i=1}^{B} X_i\right) = E\left[\left(\frac{1}{B}\sum_{i=1}^{B} X_i\right)^2\right] - E\left[\frac{1}{B}\sum_{i=1}^{B} X_i\right]^2$$

$$= \frac{1}{B^2}\left(E\left[\sum_i X_i^2\right] + E\left[\sum_{\substack{i,j \\ i\neq j}} X_i X_j\right]\right) - \frac{1}{B^2}\left(\sum_i E[X_i]^2 + \sum_{\substack{i,j \\ i\neq j}} E[X_i]E[X_j]\right)$$

$$= \frac{1}{B^2}\left(\sum_i E[X_i^2] - \sum_i E[X_i]^2 + \sum_{\substack{i,j \\ i\neq j}} E[X_i X_j] - \sum_{\substack{i,j \\ i\neq j}} E[X_i]E[X_j]\right)$$

$$= \frac{1}{B^2}\left(\sum_i Var(X_i) + \sum_{\substack{i,j \\ i\neq j}} Cov(X_i, X_j)\right)$$

$$= \frac{B\sigma^2}{B^2} = \frac{\sigma^2}{B}$$

As such if the estimators are truly independent, then as we increase the amount of estimators in the forest ($B \to \infty$), our variance will tend to 0 in the limit. However, if we make splits using a specific information criteria, the trees will be somewhat correlated, which will lead to a change in variance of the average estimator. If we take the pairwise correlation between our estimators to be ρ, this would lead to a variance of $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$

$$\text{Var}\left(\frac{1}{B}\sum_{i=1}^{B} X_i\right) = \frac{1}{B^2}\left(\sum_i \text{Var}(X_i) + \sum_{\substack{i,j \\ i\neq j}} \text{Cov}(X_i, X_j)\right)$$

$$= \frac{1}{B^2}(B\sigma^2 + B(B-1)\rho\sigma^2) = \frac{1}{B^2}(B^2\sigma^2\rho + B(1-\rho)\sigma^2)$$

$$= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

As $B \to \infty$, the variance will remain at $\rho\sigma^2$. Correspondingly, bagging will only make sense if we can ensure that the trees are created independently. In random forests this is achieved by randomly selecting variables to split on at each node. Typically people will randomly select a subset $S < F$ of features and then "greedily" pick the feature with the best information gain. From literature, random forests typically converge around 200-400 trees, and for classification forests, the general subset size is $\sqrt{|F|}$, with a minimum size of 1 [61], [62].

Whilst random forest is typically used for classification problems, a similar algorithm called isolation forest can be used for anomaly detection. Whilst anomaly detection models do exist, Many anomaly-detection models still attempt to train on the normal data without profiling the anomalies leading to extremely high false positive rates, and can be constrained to lower dimensional data due to the high computational complexity.

The isolation forest algorithm, initially described by Liu et. al in 2008 [63], [64] attempts to address both these problems by taking advantage of the characteristics of anomalies

1. They are in the minority
2. They will be far away in feature space from normal instances

Since they are so "few and different", the theory is that anomalies will be easy to isolate in a decision tree during testing. By measuring the path lengths of inputs, we will be able to effectively distinguish as to whether a data point is an anomaly (short path), or normal (long path).

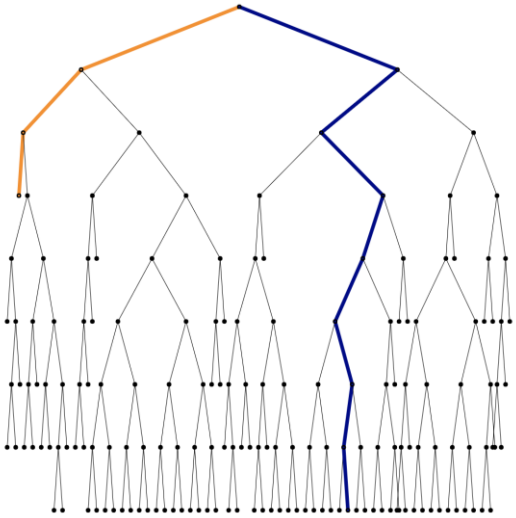


Figure 8: Decision tree path lengths for anomaly (orange) and normal data point (blue). Note how the anomaly path length is much shorter.

To understand why isolation forests do so well, we first define the issues of swamping and masking that can occur whilst attempting anomaly detection with highly imbalanced datasets. Swamping occurs when you wrongly classify anomalies as normal data points. It typically occurs when anomalies are too close to normal points in feature space, increasing the number of edges traversed through the tree to separate that instance. Masking occurs when there are high density anomaly clusters in the dataset, concealing the fact that they are anomalies as isolating these points will return long pathlengths.

Isolation forests address these issues by training estimators on small subsets of the data. This will mitigate both the swamping (Even if anomalies are close to normal points in feature space, they will be on the outskirts of the data cluster. Small sample sizes will reduce the amounts of anomalies on the outskirts, leading to shorter isolation path lengths), and masking (Smaller sample size will result in fewer anomalies in the dataset, reducing the size of anomaly clusters that may have formed).

To maintain pairwise independencies of each estimator in the forest and aggregated model variance, splits on nodes will be made at a randomly selected feature and threshold value. Due to the threshold splitting each node will have exactly two child nodes, each of which will be either an external (leaf) or internal node. This means the estimators in an isolation forest are proper binary trees. The recursive splits at nodes will continue until we can either perfectly separate the data (assuming distinct data points) or until the tree has grown to a predetermined maximum height.

The low memory requirement is now also apparent. As the worst-case stopping condition of the tree is when we can separate each datapoint we know there are n external nodes. By induction, we can prove that a tree with n external nodes will have $n - 1$ internal nodes, giving $2n - 1$ nodes in total.

Base Case:

$N = 1$, 1 ext node

$N = 2$, 1 ext node, 2 int nodes

Inductive Case:

Assume binary tree with k ext nodes has $k - 1$ int nodes

Prove that binary tree with $k + 1$ *e*xt nodes has k *i*nt nodes

Start with tree with $k + 1$ ext nodes

Remove a leaf and its sibling, now you have tree with k ext nodes

From assumption, new tree will have $k - 1$ int nodes

Add orginal leaves back in, you now have $k - 1 + 1 = k$ int nodes

This proves proposition in inductive case

This means that the number of parameters in the Isolation Forest is bounded, and scales linearly with the data ($O(n)$).

To score each data point, a unique path length is calculated. Using just the average path length to calculate the anomaly is a naïve approach as they are not normalized, and thus cannot be compared. A better approach would be a aggregation, such as the notion of expected unsuccessful path length from BST theory [65]:

$$\bar{c}(n) = 2H_{n-1} - \frac{2(n-1)}{n}$$

where $\bar{c}(n)$ can be thought of as the average path length to one of the external nodes, and H_n is the harmonic number as seen in the proof. This will scale accordingly with the path lengths, making it possible for us to normalize properly. As a note, estimated growth of tree height $log_2 n$ or average tree height could have been used as well, though it has been difficult to find an analytical answer for the latter, with a popular approximation being $\alpha \, ln \, n - \beta \, ln \, ln \, n$, where $\alpha \approx 4.31107$, and $\beta \approx \frac{3}{2 \, ln(\alpha/2)}$ [66]. The proof of this approximation is beyond the scope of this thesis. In any case, average unsuccessful path length is the normalization factor most commonly used in Isolation Forest implementations.

To bound the values of the score between 0 and 1, the following score function is used, where $E(h(x))$ is the average path length throughout the forest.

$$s(x,n) = 2^{-\frac{E(h(x))}{\bar{c}(n)}}$$

As $E(h(x)) \to 0$, $s(x,n) \to 1$

As $E(h(x)) \to \bar{c}(n)$, $s(x,n) \to 0.5$

As $E\big(h(x)\big) \to n-1$, $s(x,n) \to 0$

Anomalies will correspond to higher score values as they will have a lower $E(h(x))$. We choose $n-1$ as our upper bound because that is the maximum height a tree can reach given it is strictly binary.

When growing isolation trees, the height will be bounded by the average height of a tree given sample size n. This is because we expect anomalies to be much lower than the average tree height, so we can save on computational complexity. As no exact analytical formula exists for average height of binary trees, the original paper had decided to use an approximation made by Knuth [67] of $ceil(log_2 n)$. In most implementations, this is the approximation used to bound tree height. Note this is very close to the minimum tree height $floor(log_2 n)$, it is still justified as trees are grown on data that consists of majority "normal" points and anomalies should still be filtered out well before path lengths of $floor(log_2 n)$.

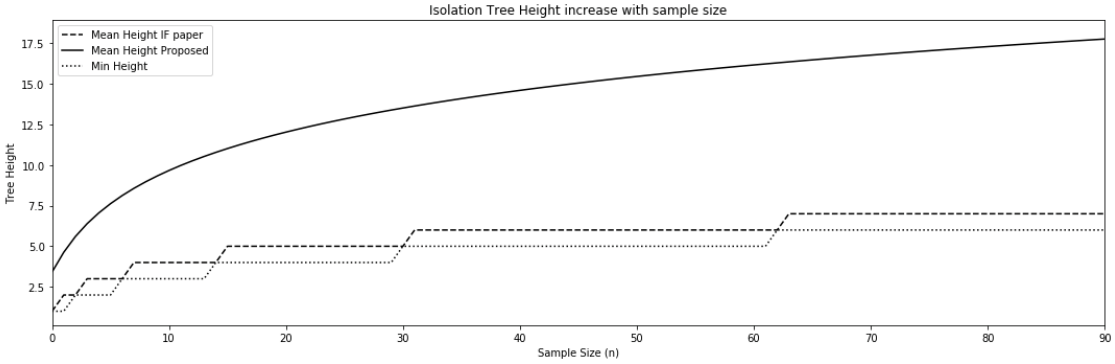


Figure 9: Increase of mean tree height as a function of sample size using 2 approximations. We provide minimum height as the lower bound.

For large sample sizes and anomalies that are close to the dataset, we feel it would be better to bound using the new average tree length. At large sample sizes with anomalies close to normal data, Knuth's approximation may prematurely declare some anomalies to be normal. This alteration will be especially relevant in future algorithms where we attempt to isolate seizures much closer to "normal" activity.

Any leaf nodes that have residual data that needs to be classified during testing can be taken as subtrees, and the average path length $\bar{c}(n)$ of that subtree can be added on during calculation of the score.

Finally isolation forest has a time complexity of $O(Bn_{samp}\, log_2\, n_{samp})$ for training and $O(NB\, log_2\, n_{samp})$ for testing, where N is the total number of test samples, n_{samp} is the subsampling size, and B is the number of estimators. This is because we cap the height limit of the trees at $log_2\, n$. Literature has shown training times of 7.6 seconds for $n_{samp} = 256$ and 11.9 seconds for $n_{samp} = 16384$ [63]. The pseudo-code to create an isolation forest is provided based on [63].

Table 7: Isolation tree algorithm

Algorithm 3: iTree($\boldsymbol{X, e, l}$)
Inputs: X – input data, e – current tree height, l – height limit **Output:** an iTree 1. **if** $e \geq l$ or $\lvert X \rvert \leq 1$: 2. **return** $ext\_node\{Size \leftarrow \lvert X \rvert\}$ 3. else: 4. let Q be a list of features in X 5. select random feature in $q \in Q$ 6. find max and min values of q, then uniformly sample threshold value p 7. $X_l \leftarrow$filter all elements in X where $q < p$ 8. $X_r \leftarrow$filter all elements in X where $q \geq p$ 9. **return** $int\_node\{L \leftarrow iTree(X_l, e+1, l), R \leftarrow iTree(X_r, e+1, l),\ split_{feat} \leftarrow q, split_{value} \leftarrow p$ 10. end if

Table 8: Isolation Forest algorithm

Algorithm 4: iForest(X, $\boldsymbol{t, \psi}$)
Inputs: X – input data, t – number of trees, ψ – subsampling size **Output:** a forest of t iTrees 1. **Initialize** empty set $Forest$ 2. set height limit $l = ceil(\log_2 \psi)$ 3. **for** i = 1 to t: 4. $X' \leftarrow subsample(X, \psi)$ 5. $Forest \leftarrow Forest \cup iTree(X', 0, l)$ 6. end for 7. **return** $Forest$

Table 9: Isolation Forest path length algorithm

Algorithm 5: PathLength($\boldsymbol{x}, \boldsymbol{T}, \boldsymbol{e}$)

Inputs: x – a test data point, T – an iTree, e – current path length
Output: path length of x

1. initialize $e = 0$
2. **if** T is external node:
3. **return** $e + \bar{c}(T.size)$
4. end if
5. $a \leftarrow T.split_{feat}$
6. **if** $x_a < T.split_{value}$:
7. **return** $PathLength(x, T.L, e + 1)$
8. **else if** $x_a \geq T.split_{value}$:
9. **return** $PathLength(x, T.R, e + 1)$
10. end if

Extended Isolation Forest

In isolation forests, branch cuts are made on random thresholds on random features. It can be visualized as a decision tree, but alternatively it can also be thought of as separating the points with hyperplanes in feature space. As an example if we take a dataset $\boldsymbol{Z} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where x and y are features of each data point. Each isolation tree is grown by randomly picking one of those features on each iteration, and then randomly make a cut on a random threshold in that feature. This amounts to randomly drawing orthogonal hyperplanes in your feature space.

In general, these cuts will happen more often in places where the data is greatly clustered as the isolation forest attempts to isolate each data point. As the branches are constrained to be made on features, there will be concentrated regions in space that have a lot of cuts through them despite containing no data points. When averaging over many such trees this will create artefacts in the decision function, biasing the algorithm into classifying points in these regions as normal when they are not. An example is shown below on a sample data set adopted from the work of Hariri et. al [68].

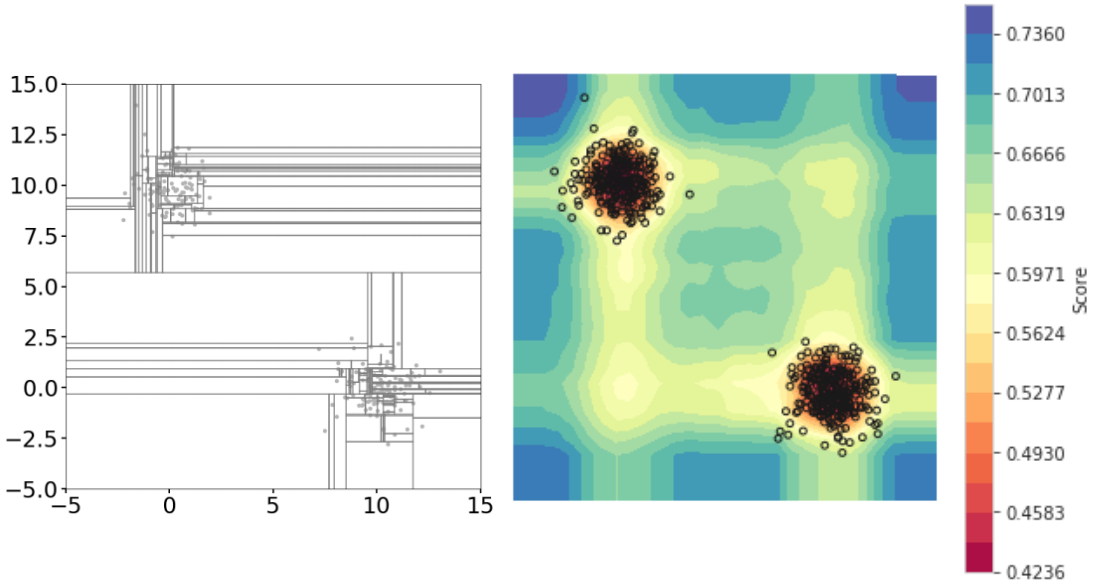


Figure 10: Vanilla Isolation Forest implementation on a toy dataset showing presence of artefacts. a) shows the cuts for a single isolation tree. Note the perpendicular cuts extend to interfere with the space in the opposing corners. b) shows the resulting score map over all values in the space. Note the presence of highly normal regions in the upper right and bottom left corners that should be classified as anomalies.

In order to combat this phenomenon, extended isolation forests branch across features, effectively choosing a random hyperplane in the vicinity of the dataset to cut on [68]. With this strategy, there will never be any concentrated focusing of branches in any region of the feature space except in areas of high data density. Implementing the extended isolation forest with full extension on the original data shows a more intuitive determination of the data distribution, extending radially from the high-density regions adopted from Hariri et. al [68].

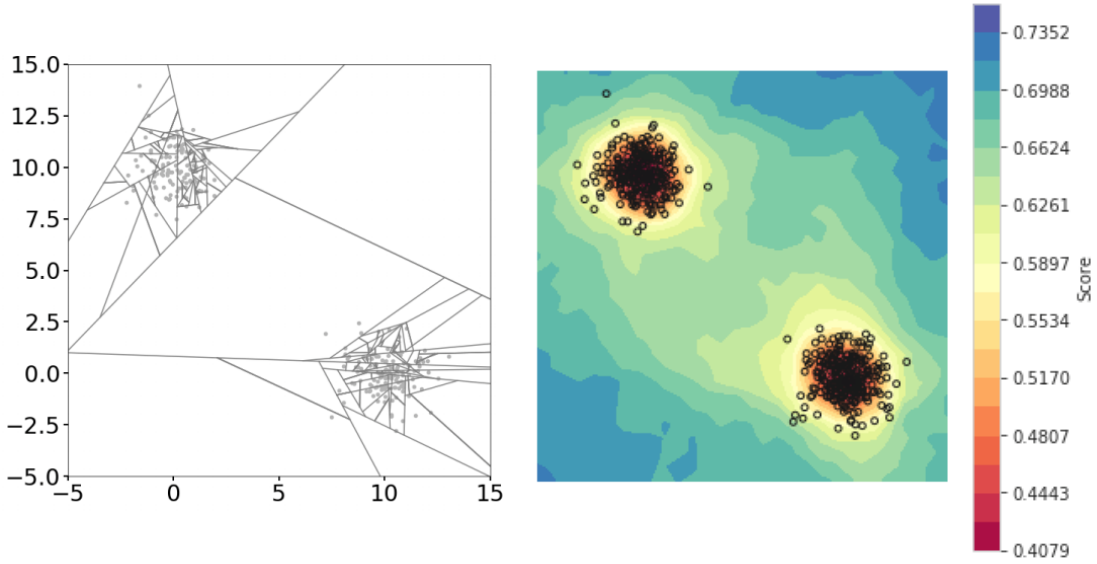


Figure 11: Extended Isolation Forest implementation on a toy dataset. a) shows the cuts for a single extended isolation tree. Note there is no concentration of cuts in any region except around the datapoints. b) shows the resulting score map over all values in the space. Notice how anomaly scores now extend radially around the datasets as we would expect.

To determine the hyperplane, we require the slope value $\boldsymbol{w}$ and the bias value $\boldsymbol{b}$. This is similar to the SVM setup where we define the equation of the hyperplane. We constrain $\|\boldsymbol{w}\| = 1$, and randomize it's direction by selecting its value randomly over a D-dimensional unit hypersphere, where D is the dimensionality of the feature space. During implementation this can be achieved by a number of methods including rejection sampling [69], trigonometry method, and coordinate method[69]. A more elegant method is to sample $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, where $\boldsymbol{I}$ is the identity matrix.

Proof: First let $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. For any orthogonal matrix $\boldsymbol{Q}$, $\boldsymbol{QX} \sim \boldsymbol{\mathcal{N}}(\boldsymbol{0}, \boldsymbol{I})$. This is because of the property of orthogonal matrices $\boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{Q}\boldsymbol{Q}^\top = \boldsymbol{I}$. Additionally, as $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ has an inner product in the exponent, $(\boldsymbol{QX})^\top \boldsymbol{QX} = \boldsymbol{X}^\top \boldsymbol{Q}^\top \boldsymbol{QX} = \boldsymbol{X}^\top \boldsymbol{X}$. As orthogonal matrices are unitary transformations, the distribution of $\boldsymbol{X}$ is rotationally invariant, and is only dependent on the length $(\boldsymbol{X}^\top \boldsymbol{X})$. To limit the length, we set $\boldsymbol{Y} = \frac{\boldsymbol{X}}{\|\boldsymbol{X}\|}$. Since we have already shown that $\boldsymbol{X}$ is invariant to rotations, so is $\boldsymbol{Y}$. $\boldsymbol{QY} = \frac{\boldsymbol{QX}}{\|\boldsymbol{QX}\|} = \frac{\boldsymbol{QX}}{\|\boldsymbol{X}\|}$.

Since $\boldsymbol{Q}$ is orthogonal, $\|\boldsymbol{QY}\|$=1. Hence as we have shown rotational invariance whilst maintaining length, proving that we are uniformly sampling from a unit sphere.

During implementation, each component of $\boldsymbol{w}$ can be sampled independently from $\mathcal{N}(0,1)$. To choose the bias $\boldsymbol{b}$ we sample each component uniformly. In order to ensure we aren't slicing too far from the dataset, we select from each of the available feature ranges. We finally define our plane as the set of all points that satisfy

$$(\boldsymbol{x} - \boldsymbol{b}) \cdot \boldsymbol{w} = 0$$

To make our branches, we just cut on this hyperplane all points that satisfy $(\boldsymbol{x_i} - \boldsymbol{b}) \cdot \boldsymbol{w} > 0$ will be passed to the right subtree, whilst all other points will be passed to the left.

Extension Levels

It is noted that in the example above with a 2D feature space, a vanilla isolation forest can be made by applying a random binary mask to $\boldsymbol{w}$ and then renormalizing [68]. The hyperplane is then normal to either one or the other feature, where the bias value will act as the threshold. This can be extended to n-dimensions, as it is always possible to apply a random binary mask where $N - 1$ components are 0. This is defined as Extension Level 0. In n-dimensional feature space, we can extend this concept by applying binary masks with up to $N - 1$ randomly chosen components components being 0. As such we can go from 0 to $N - 1$ extension levels, defining the amount of features to cut across when defining our random hyperplane.

In the case of data that has equal variance in all feature dimensions, the fully extended method will reduce the artefacts produced from the random splitting at lower levels. However if there is low variance in certain dimensions, it may be better to reduce the extension level to reduce computational overhead. To

incorporate extension levels, the user can be asked to provide an extension number N_{EX} during training. A random binary mask can then be made with $N - N_{EX}$ 1's, then take the Hadamard product with $\boldsymbol{w}$.

The differences with the standard isolation forest mainly pertain to how the trees are generated. The hyperplane splitting method is added, as is the binary mask for the extension level. Otherwise, everything else (Bagging, Score function), remains the same [68].

Table 10: Extended Isolation tree algorithm

Algorithm 6: Extended iTree($\boldsymbol{X, e, l, ext}$)

Inputs: X – input data, e – current tree height, l – height limit, ext – extension level
Output: an iTree

1. **if** $e \geq l$ or $|X| \leq 1$:
2. **return** $ext\_node\{Size \leftarrow |X|\}$
3. else:
4. randomly select a normal vector $n \in \mathbb{R}^{|X|}$ from a unit hypersphere
5. Randomly select an intercept vector $b \in \mathbb{R}^{|X|}$ inside the extrema values of each individual feature
6. Randomly set ext coordinates of n to 0
7. $X_l \leftarrow$filter all elements in X where $(X - b) \cdot n \leq 0$
8. $X_r \leftarrow$filter all elements in X where $(X - b) \cdot n > 0$
9. **return** $int\_node\{L \leftarrow iTree(X_l, e + 1, l), R \leftarrow iTree(X_r, e + 1, l),\ Normal \leftarrow n, intercept \leftarrow b$
10. end if

Table 11: Extended Isolation Forest path length algorithm

Algorithm 7: PathLength($\boldsymbol{x, T, e}$)

Inputs: x – a test data point, T – an iTree, e – current path length
Output: path length of x

1. initialize $e = 0$
2. **if** T is external node:
3. **return** $e + \bar{c}(T.size)$
4. end if
5. $n \leftarrow T.Normal$
6. $b \leftarrow T.Intercept$
7. **if** $x_a(X - b) \cdot n \leq 0$:
8. **return** $PathLength(x, T.L, e + 1)$
9. **else if** $(X - b) \cdot n > 0$
10. **return** $PathLength(x, T.R, e + 1)$
11. end if

Implementation (Cross Validation and Tuning)

Evidence Accumulation (Smoothing) Filter

All of the anomaly detection techniques discussed in this section output a hard class label of 0 or 1, depending on whether it thinks a data-point is an anomaly. To smoothen the output, we pass all detection values through a first order Infinite Impulse Response (IIR) filter. This is a real-time low pass filter that will make the detector more robust to any transient anomalies. The detector will only conclude that a seizure is occurring after enough consecutive evidence has been accumulated. This robustness comes at the trade-off with latency. This is an acceptable trade-off in our application, as we have determined a clinical window of approximately 1 minute after onset within which to alert the user. The filter will have a user-defined hyperparameter α which will also have to be tuned for during cross-validation. Define $x(t)$ to be the output of a classifier at time t. Define $y(t)$ to be the output of the filter.

$$y(t) = \alpha x(t) + (1 - \alpha) y(t - 1)$$

Note that we will still need to define a threshold. This will be determined from the ROC curve after optimal hyperparameters have been obtained.

To determine the ROC curve during one of the cross-validation folds, we first run the outputs of an algorithm that has classified seizure data through the smoothing filter. An example of algorithm output (black) and smoothing filter output (red) is shown for 10 seizures.

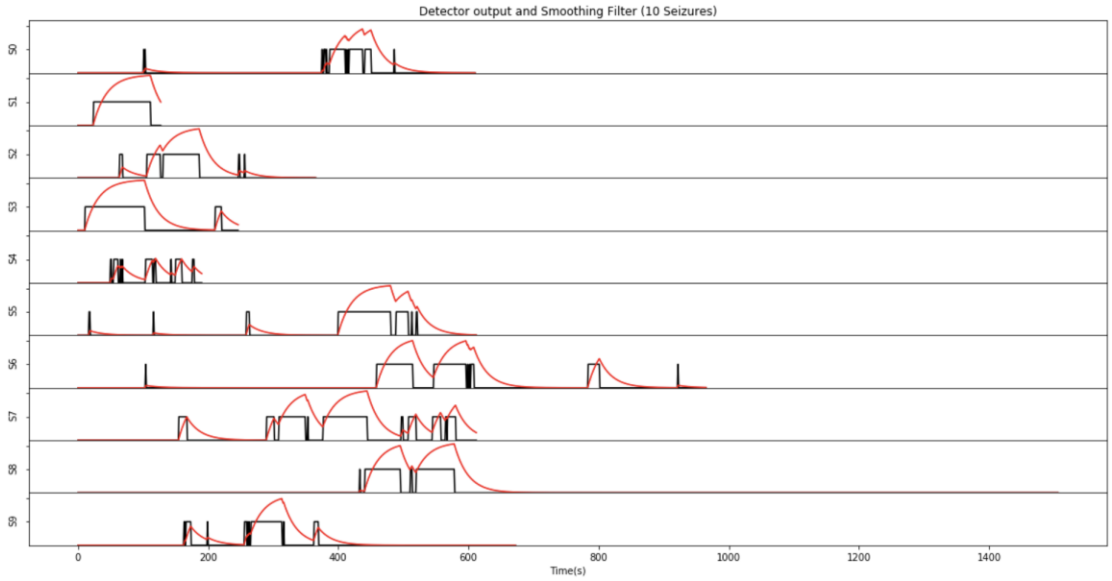


Figure 12: Anomaly detector output with accumulation filter over 10 seizures with the binary detection outputs (black) and corresponding accumulation filter outputs (red) are provided.

The maximum value of the smoothed detections for each detection is calculated and stored in an array. We then set our initial threshold slightly below the minimum value of this array, guaranteeing 100% sensitivity of our detector. The amount below the minimum value is set by a user-defined parameter $\Delta_t = 0.025$.

Our maximum threshold value will be similarly set by running all the non-seizure samples through the algorithm and smoothing filter, then calculating the maximum value reached, and setting the threshold Δ_t above it. This guarantees 0% False Positive Rate.

We then test then test the detector for all thresholds between the min and max, separated by Δ_t. This covers the entire ROC curve. If thresholds are too far, this interval may be increased to save on time. Note when testing the detector, the algorithm outputs are run through a custom filter prediction function that

incorporates a 10-minute refractory period after detection. No detections can occur in this time. 10 minutes was decided upon after consulting with physicians.

Performance Metrics

The performance metrics for each anomaly detection algorithm are sensitivty, false positive rate (FPR) and latency. An ROC curve can be generated by varying the threshold as done previously.

FPR is calculated by

$$\frac{\text{detections in dataset}}{\text{length of dataset in days}}$$

Sensitivity is calculated by

$$\frac{\text{seizures detected}}{\text{total seizures}}$$

The Area Above ROC (AAROC) value can then be calculated as a performance metric by using numerical integration with the composite trapezoidal rule. As we are expressing FPR in days, AAROC value will have units (/day). A smaller AAROC value corresponds to a better detector. Finally we also calculate a latency value defined as the time difference between the clinical onset of the seizure (set by physician) till detection.

Dataset Split

At the time of training, we had approximately 1630 total usable hours of data, determined from the number of blocks obtained after the preprocessing and feature extraction steps. We want to randomly separate this data into equal training and testing sets, whilst maintaining the temporal integrity of each segment.

This is done by initializing a training and testing array, and appending random segments from the dataset without replacement to either array. Every time a segment is appended, the length of each array is recalculated. The next segment is appended to the smaller array. In this way, we end up with a training and testing array of approximately 815 hours each.

In order to test the model appropriately, a good estimation of the false positive rate must be determined. We will calculate the false positive rate per day by running the detector through a contiguous array of these data segments, simulating a user wearing the watch in the real world.

n-fold cross-validation is performed entirely on the training dataset. We split the training dataset into n folds by randomly choosing segment indices and appending then to an array until the desired length of $\frac{N}{n}$ is reached, where N is the total length of the training dataset. This is performed n times, once for each fold. During cross validation, all segments not indexed in the fold (length $N - n$) are used to train the classifier, whilst the segment within the fold (length n) is used to test the classifier. Once the optimal hyperparameters have been selected from the grid search, we retrain and then test the algorithm using these parameters on the original train/test split.

Results

Offline Anomaly Detection

For each model, 4 graphs will be presented. The first will be the grid-search graph, where each pairwise parameter selection will have an averaged AAROC over 5 fold cross-validation. The selected hyperparameters are then used to train a new classifier, and 3 other graphs show the performance characteristics of this final classifier. The table will show performance characteristics for the classifier created during the grid-search.

OCSVM

Initially, the quadratic program was solved using a numerical QP solver in Python (see SVM implementation section). Whilst it gave correct results, it was an unoptimized implementation. For faster results in cross-validation and ROC curve calculations, the sci-kit learn library was used. This library has the same problem formulation implemented in C, avoiding the slow and memory intensive Python Interpreter.

Cross validation was performed over with $n = 5$, leading to testing folds of approximately 160 hours. To prevent any one feature from dominating over any another, we perform standardized scaling (z-score normalization) to the data before training and testing. This feature scaling is especially important as we are using the RBF kernel.

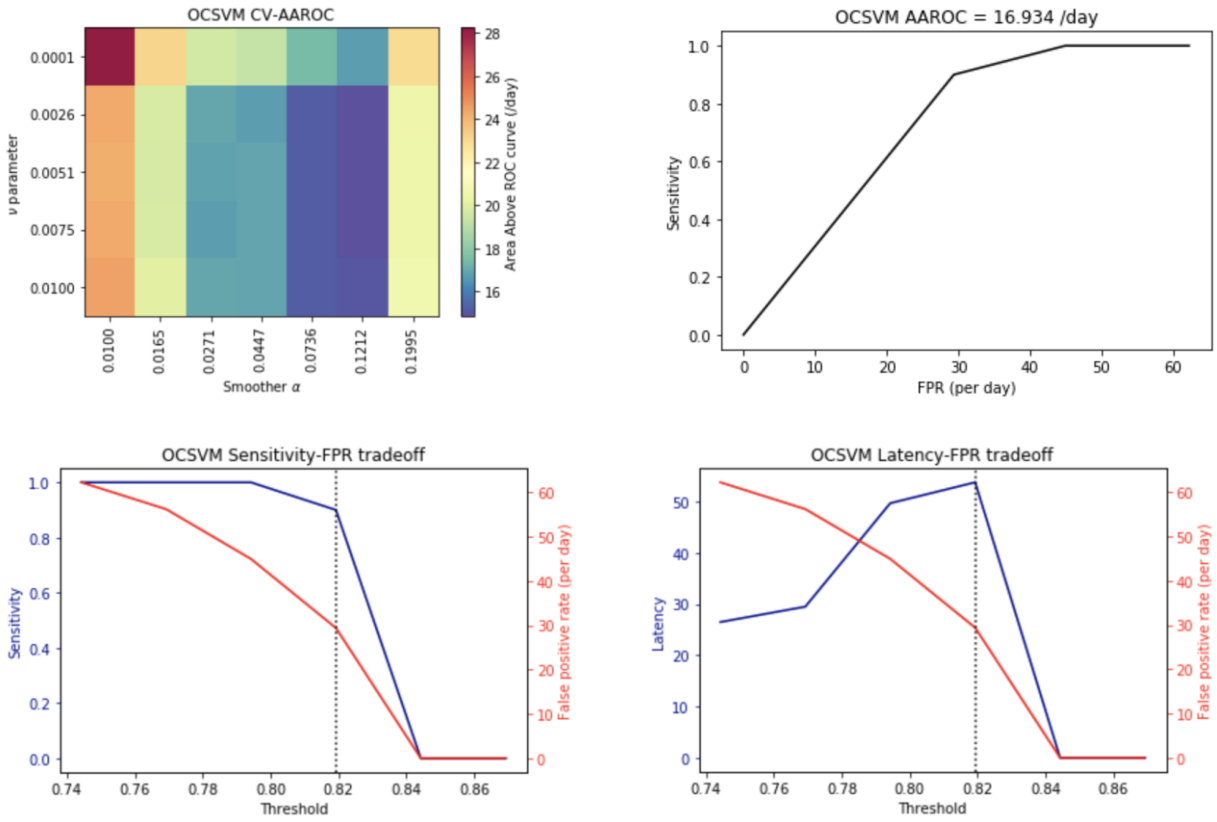


Figure 13: Results of OCSVM. a) shows the grid-search and cross-validation results. b) shows the ROC curve at the optimal hyperparameters. c) shows the sensitivity-FPR tradeoff. d) shows the latency-FPR tradeoff

Table 12: OCSVM optimal performance characteristics (grid search)

Optimal nu	0.0031623
Optimal alpha	0.1211528
Optimal AAROC	14.8608/day

OCSVM was used as a baseline to measure our worst performance. We note that whilst sensitivity and latency are acceptable, the false positive rate at almost 29.3/day is too high for our required performance characteristics. A reason for this may be because we could not use the entire dataset to perform training on OCSVM as it was too big for the matrix multiplications to handle. Instead we randomly selected $\frac{1}{5}$ of the dataset (1×10^6 samples). It is possible that many false positives were missed during training, which would correspond to these characteristics. We expect a similar performance for SVDD. In future iterations

of the algorithm, it may be better to filter data using another algorithm that can train with the entire dataset (forest-based methods), and then use this selected subset to train the SVM methods.

SVDD

No SVDD library currently exists for Python, so we built a custom SVDD implementation. It was designed to mirror the functionality of the OCSVM sci-kit learn implementation, without Cython optimizations. As mentioned in the SVM implementations section, the **cvxopt library** was the QP solver used to solve the dual optimization problem.

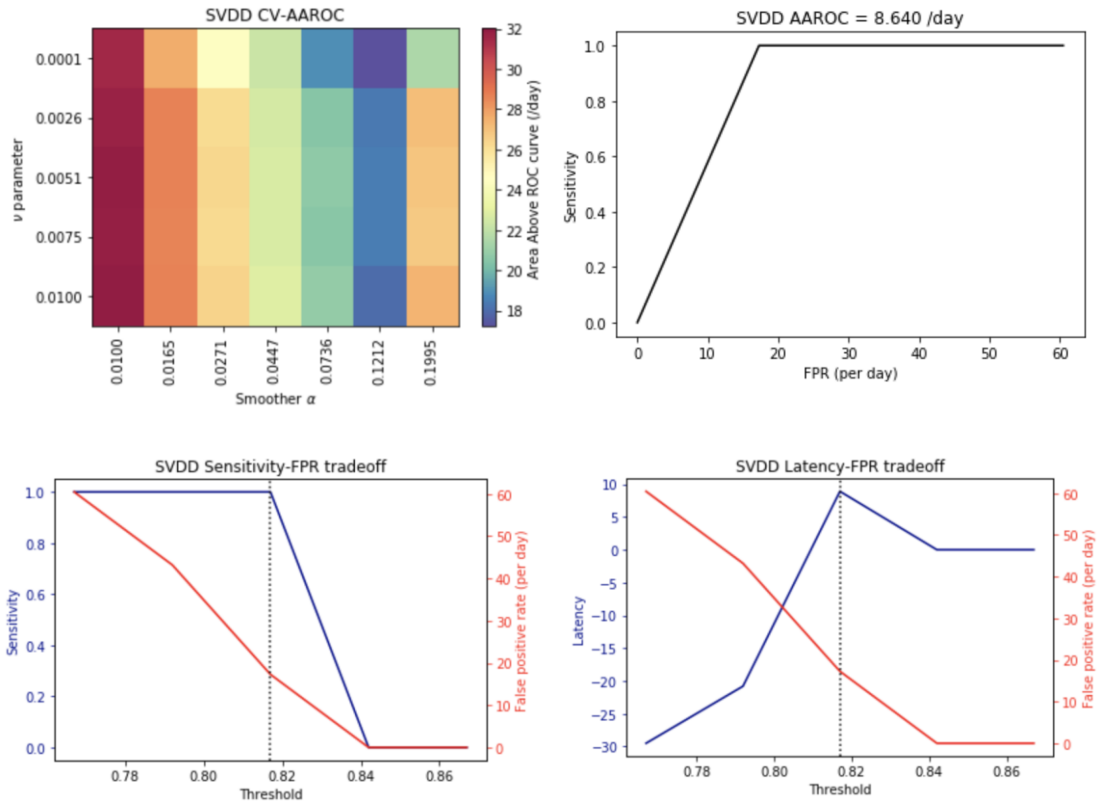


Figure 14: Results of SVDD. a) shows the grid-search and cross-validation results. b) shows the ROC curve at the optimal hyperparameters. c) shows the sensitivity-FPR tradeoff. d) shows the latency-FPR tradeoff

Table 13: SVDD optimal performance characteristics (grid search)

Optimal nu	0.0001
Optimal alpha	0.1212
Optimal AAROC	8.640

The SVDD latency is a very at 8.9 seconds with the optimal detector. Notice on the latency chart that there are negative numbers. This means that the seizure was detected before our clinical start marker. Generally this can only occur if the detector is classifying too many points as seizures. If this is the case, we should see a proportionally high false positive rate. Interestingly, the false positive rate is at 18.4/day, at 100% sensitivity. This performance is far superior than the OCSVM. Though the false positive rate is still too high, we can attempt to tune it in future iterations by sacrificing some latency. SVDD seems like a method we should investigate in future development.

IF

We used the native sci-kit learn implementation of isolation forest. This provided an optimized implementation of the algorithm, with other convenience parameters that we could tune. We chose 200 estimators as recommended in literature. We wanted to give the forest an opportunity to see all the data, and thus chose subsampling size according to

$$floor\left(c_s \times floor\left(\frac{length\ of\ total}{number\ of\ estimators}\right)\right)$$

where c_s is the detector sampling factor. We set this value to 1.2. This leads to a subsampling size of approximately 14,000 during cross-validation.

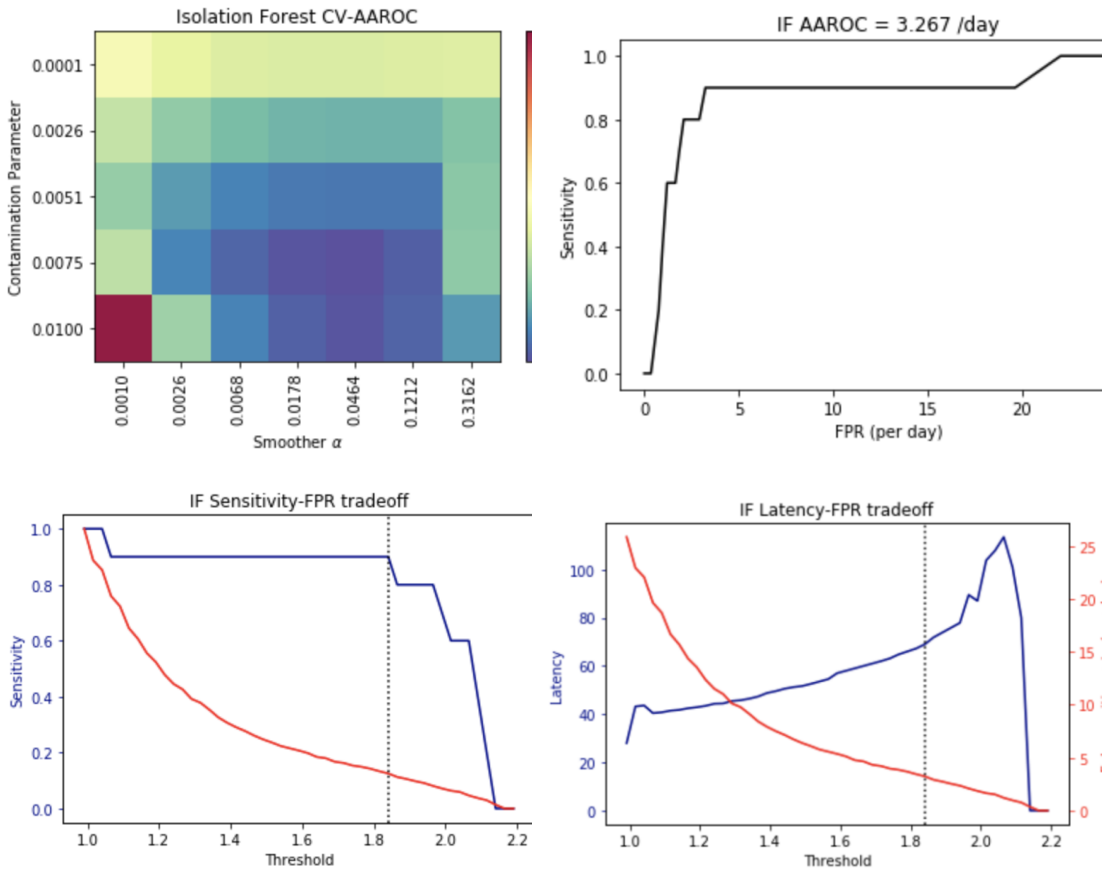


Figure 15: Results of Isolation Forest. a) shows the grid-search and cross-validation results. b) shows the ROC curve at the optimal hyperparameters. c) shows the sensitivity-FPR tradeoff. d) shows the latency-FPR tradeoff

Table 14: Isolation Forest optimal performance characteristics (grid search)

Contamination	0.0075
Optimal alpha	0.0464
Optimal AAROC	3.1018/day

With an AAROC of best classifier in terms of both sensitivity and false positive rate by a margin. It has a high latency of just over 60 seconds, but that is just on the edge of our clinical window. We can decrease the threshold with a tradeoff of a higher false positive rate if require faster detection.

Extended IF

The extended isolation forest algorithm was implemented following [68]. The corresponding changes were made to the isolation tree and the score function. Extension level was maximized at 8. The same subsampling size as the isolation forest was used.

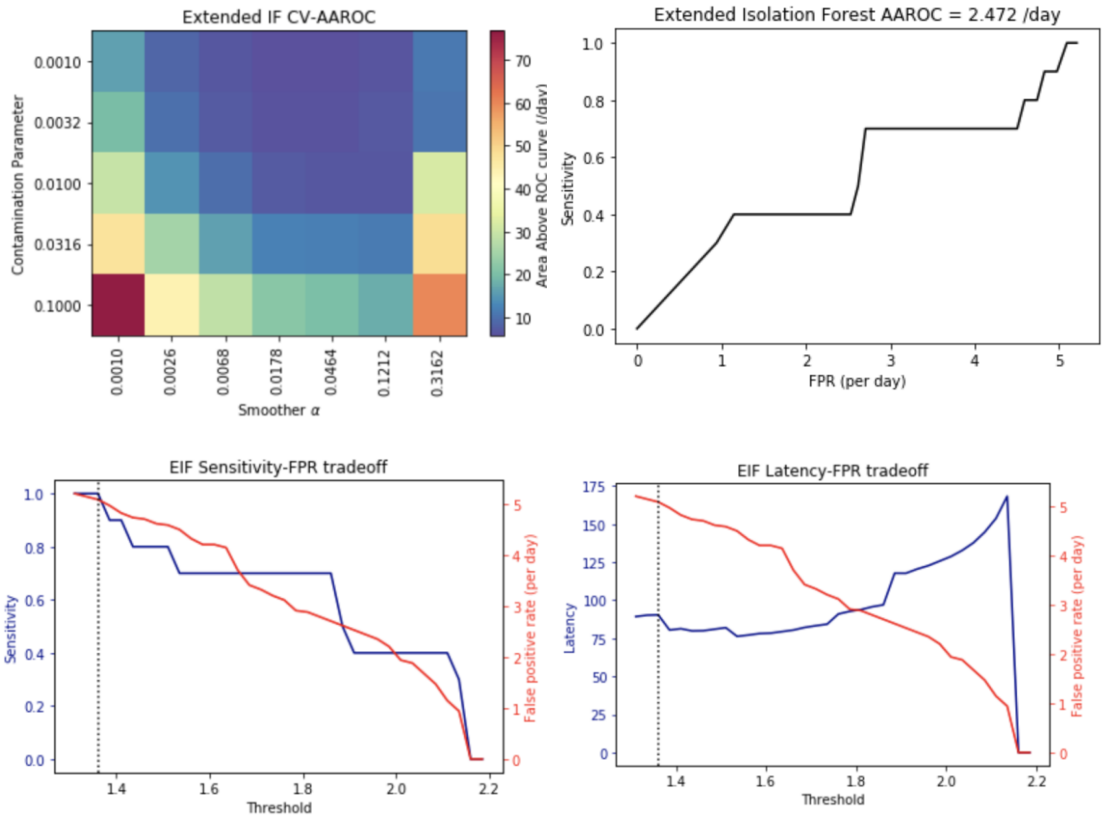


Figure 16: Results of Extended Isolation Forest. a) shows the grid-search and cross-validation results. b) shows the ROC curve at the optimal hyperparameters. c) shows the sensitivity-FPR tradeoff. d) shows the latency-FPR tradeoff

Table 15: Extended Isolation Forest optimal performance characteristics (grid search)

Contamination	0.0010
Optimal alpha	0.0178
Optimal AAROC	5.8061/day

This is perhaps the most surprising result. We expected the extended isolation forest to perform better than the isolation forest, but that does not seem to be the case. Whilst it may provide a higher sensitivity, it provides a much higher false positive rate and a latency more than double that of the Isolation Forest. The high latency makes sense as the optimal alpha value is more than half that of the isolation forest. A possible reason for why the false positive rate is so high is seen when we attempt to run the seizures through the EIF and compare to IF output. It seems EIF is more sensitive to the features compared to the isolation forest. This is a result of the chosen parameters, and it could be that our grid search was not wide enough. It still outperforms the support vector classifiers in terms of both false positive rate and latency.

Summary

Table 16: Summary of optimal anomaly detectors

Model	Sensitivity	FPR (/day)	Latency (s)
OC SVM	0.9	29.3	53.7
SVDD	**1.0**	17.28	**8.9**
IF	0.9	**3.2**	69.0
EIF (Ext = 8)	**1.0**	5.0	90.4

Upon looking at the results, the isolation forest model was selected to be implemented on the Apple Watch due to the low false positive rate and latency a latency near the clinical window. SVDD was a possible choice, but it is computationally complex in comparison, and implementation would require careful optimization. Additionally, the isolation forest algorithm has a much better false positive rate, though admittedly the latency is markedly worse.

Real-Time Anomaly Detection

Implementation

To port the model all of the nodes, children, and threshold values for each iTree are placed in a JSON file that can be saved inside the application.

Then the structure of each tree was rebuilt in Swift, using the values saved inside the JSON file. All other relevant parameters like filter coefficients, threshold, window lengths, etc. are also included in this JSON file. In this way, any time we retrain the detector, we do not have to alter the Swift implementation.

Summary

Once implemented, the detection algorithm was run for approximately 10 months in the EMU for validation. All validation metrics were collected from EMU data, though beta users also used the app to see FPR in ambulatory environments.

In Apple Watch Series 1, EMU sensitivity was approximately 90%, whilst the False Positive Rate was approximately 2/day. We say approximately as due to large data drops, it was often difficult to determine if a False Negative was due to the algorithm or due to lack of data.

After transitioning to Series 3/4, the data drop issue was reduced. Over 2000 hours tracked after transitioning, we recorded a sensitivity of 100% with a FPR of 1.29/day with 24 seizures detected. It is noted that false positives often closely mimicked seizure activity, and were commonly caused by activities involving oscillatory hand motion (washing hands, tapping, clapping, waving etc.). Additionally, despite being highly informative, it is still unclear how much heart rate features affect detection. Many false positives occurred with a constant resting heart rate, though some (especially detections where the patient had stood up) had a small heart rate increases due to homeostasis.

False positive rates for ambulatory users was an average of 3 per day. Common activities triggering the detector were driving, running and weightlifting. The latter two activities are more challenging due to heart rate increase, but it is noted that they lack a descending chirp in the frequency domain.

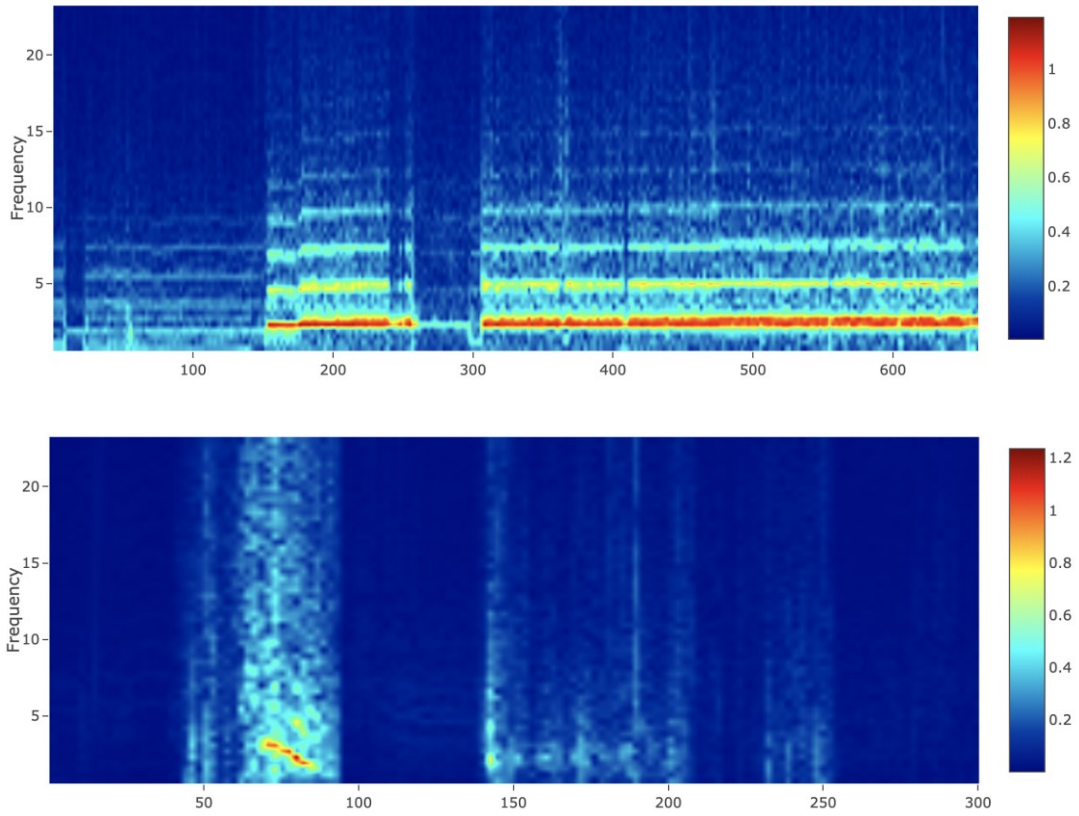


Figure 17: Spectrogram of false positive and seizure. a) shows the false positive. Notice the constant frequency pattern and the corresponding harmonics. b) shows a seizure. Notice the descending 'chirp' characteristic.

Hybrid Model (2nd Stage)

Current state-of-the-art watch-based seizure detection systems show comparable results to the Isolation Forest algorithm. Empatica's Embrace is one such system, with a recent publication showing results of 100% sensitivity and an FAR of 0.4/day [70] during an inpatient study with 135 patients (40 seizures), though they have not released any information on the latency of their system.

According to a comprehensive user and physician survey [13], the maximum acceptable false positive rate for seizure detection systems is 0.14/day, or 1 false positive per week. An ideal system would have one false positive for every true positive [13]. This is hard to quantify as seizure frequencies vary significantly per patient, but we estimate it to be approximately 1 false positive every month (0.03/day).

There are at least two areas of the isolation forest algorithm which can be improved upon

1. Our selected features are suboptimal for the task of distinguishing between GTCS and False Positives.
2. The isolation forest algorithm does not consider the temporal evolution of the seizure.

Noting that dataset imbalance is mitigated if we are separating between IF false positives and seizures, our proposed strategy is to create a classifier that distinguishes between seizures and false positives after the original detector has identified an anomaly. This is similar to model stacking, except here the original detector acts as an indicator for the 2nd stage.

To address the 2 issues discussed above, we decided neural networks would be an ideal solution. During training, deep learning models learn features that best separate the provided classes. We limit model selection to those with long-term memory, ensuring that temporal information is regarded during classification. The three models considered are

- **Long Short-Term Memory (LSTM) Network**
- **Temporal Convolutional Network (TCN)**
- **CNN-LSTM**

Preprocessing

Offline Pipeline

All anomaly detections are stored as a timestamp in the backend. A dataset of false positives is generated by pulling a 10-minute data segment around the detections (4 minutes before, 6 minutes after). Any data overlap conflicts are handled by taking the value with the earlier timestamp. All pulled false positive data is from EMU users only. Data from the beta users will be used in future iterations of the algorithm for ambulatory detection.

We stagger the segment as detection time is not centered within the high activity region. As some seizures subtypes mimic characteristics of GTCS, we remove all other seizure types (FUS, Hypermotor) from the false positive data. All GTCS segments were selected after validation by a physician using a video EEG and pulled as 10 minute segments. These segments were further fine-tuned to include only the tonic clonic portion, whose durations varied from 48 seconds to 3 minutes, demonstrating the variability of seizure lengths. A seizure segment and fine tuning window are shown in the Figure below.

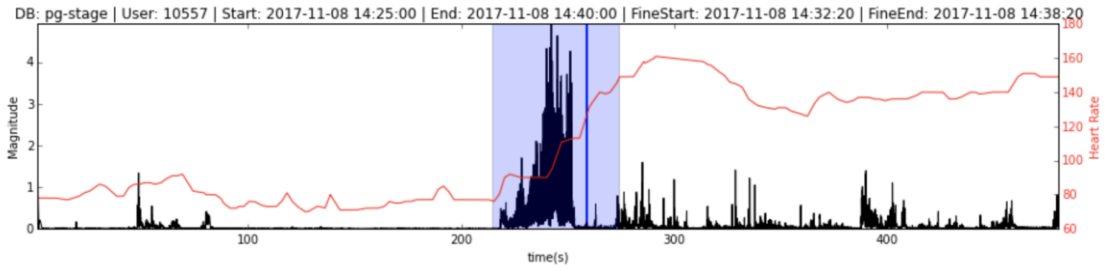


Figure 18: Seizure segment showing entire window and fine window. Blue line corresponds to isolation forest detection time.

All of the data segments are then interpolated using linear interpolation at 100Hz. Linear interpolation was chosen instead of uniformization because heart is generally slow varying, and is more likely to make

the gradual increase shown in linear interpolation rather than the steep jumps with uniformization. Unless there was a samples present, the edges of the windows were padded using nearest neighbour interpolation.

Other interpolation methods such as simple and exponential moving averages were experimented with, but eventually disregarded. Simple Moving averages lag too far behind the heart rate, whilst the exponential moving averages seem to approximate linear interpolation. A number of these averages can be seen in Figure 19. Additionally, there is an optimized vDSP framework for Swift which natively implements Linear interpolation, simplifying algorithm implementation.

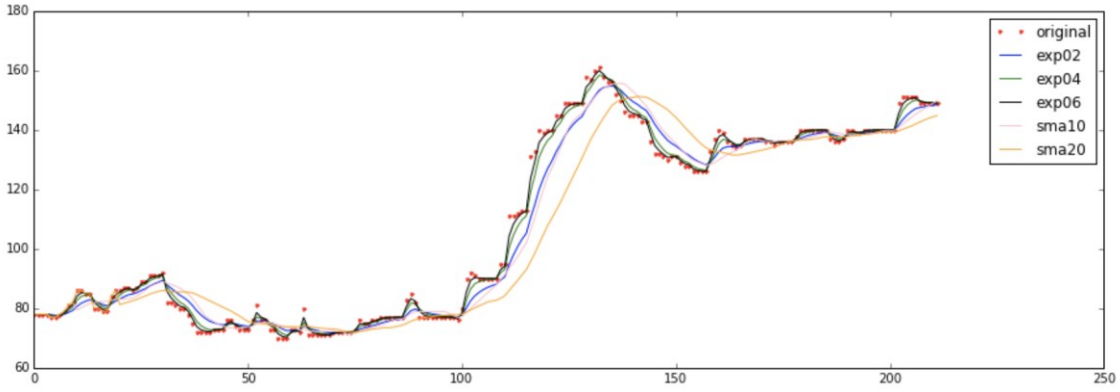


Figure 19: Filter comparison for heart rates. We have compared exponential average filters with α = 0.2, 0.4, and 0.6, as well as simple moving average with window sizes of 10 and 20 samples

Filtering for the accelerometer data is performed on all segments, using a digital high pass filter with a cutoff frequency of 0.5Hz (2^{nd} order Butterworth) to remove the gravitational effect as well as any other low frequency trends. A low pass filter with a cutoff frequency of 20Hz (4^{th} order Butterworth) was also used to any remove high frequency noise and spiking artefacts. This filter was IIR (Transposed-Direct-Form II Structure), applied in one direction. The heart rate data was not filtered.

At the time of training, there were 192 useable false positives stored in the backend, alongside 22 validated seizures. This amounted to 31 hours of false positive data and 36 minutes of seizure data. To allow enough time to understand the seizure temporal characteristics, we decided to use a sliding window size of 45 seconds (4500 samples). Seizures can last up to 90 seconds,\]but specific characteristics like the chirp signal can still be captured within this 45 second window. The overlap of consecutive windows was uniquely determined for both seizure (99% overlap) and false positive data (90% overlap) such that there was a balanced number of windows (approximately 25000) for each class (oversampling). Initially it was thought that the high overlap and relatively small dataset would cause any model to overfit. However during preliminary testing it was discovered there is good generalization if the network is shallow with appropriate regularization.

To center the data and ensure similar scaling of the features we performed z-score standardization on each window, with the standardization parameters derived over the entire dataset. Note that standardization is not strictly necessary for all neural networks. In any network that exclusively contains linear operations of the input, rescaling of the input vector can effectively be undone by changing the corresponding weights and biases.

Standardization is used because it provides better weight initializations and faster convergence during backpropagation using gradient descent. After standardization, all features will be 0 centered. As initial weights and biases in neural networks are selected to be small values (all networks in this thesis initialize weights and biases uniformly between -0.05 and -0.05), is likely that the initial hyperplane dictated by these weights pass close to the origin.

If the data was not centered, these initial hyperplanes will likely miss the data entirely, significantly affecting training speed. The primary disadvantage of standardization is alteration of the original dataset,

reducing the information available for the network to make a decision. As a workaround you could transform the initial weights rather than the inputs, but this is more involved.

All the standardized data is then used to train the neural networks. During testing, we attempt to replicate the online pipeline, and pass data in chunks of 45 seconds with a 5 second lag. The entire sequence will be passed to find the maximum value the accumulation filter reaches, and that value will be used in cross validation.

Online Pipeline

We save the standardization values in the JSON file containing all other parameters that is uploaded to the watch. An additional running buffer of 2250 samples (45 seconds x 50 Hz) is implemented to store 45 seconds worth of data. This buffer will constantly update as new data comes in, until the isolation forest detector detects an anomaly. At that point, the buffer is passed through a linear interpolation function (**vDSP_vlint**), after which 45 seconds worth is taken. Should the data not cover 45 seconds, the edges are handled using nearest neighbor interpolation. The interpolated data is filtered using identical coefficients to the offline case, standardized, then passed to the neural network for classification. The data in the running buffer will be updated and passed to the network in 5 second lag intervals for 2 minutes after Isolation Forest detection. This lag interval is required to remain under the 15% CPU constraint. All outputs are then passed through the evidence accumulation filter which will alert the user if a specified threshold is passed within the 2-minute time-frame. If not, the detection is assumed to be a false positive, and the entire pipeline is reset.

Theory (2nd Stage)

Recurrent Networks

Recurrent Neural Network

In a traditional neural network, we assume that all of the inputs and outputs are independent of one another. This is not always true, with common exceptions coming in language modelling. There is a similar dependence of inputs in seizure data, leading to the need of a strategy that can encode the temporal dependencies of the input. Recurrent neural networks can do this by encoding the memory of all the inputs that have been calculated so far. We provide a small example below. Let $\boldsymbol{X} \in \mathbb{R}^{4\times4500}$ be an input window to our network. Let $x_t \in \mathbb{R}^{4\times1}$ be a single data point with x, y, z and HR features at time t. An RNN could then be thought of as follows.

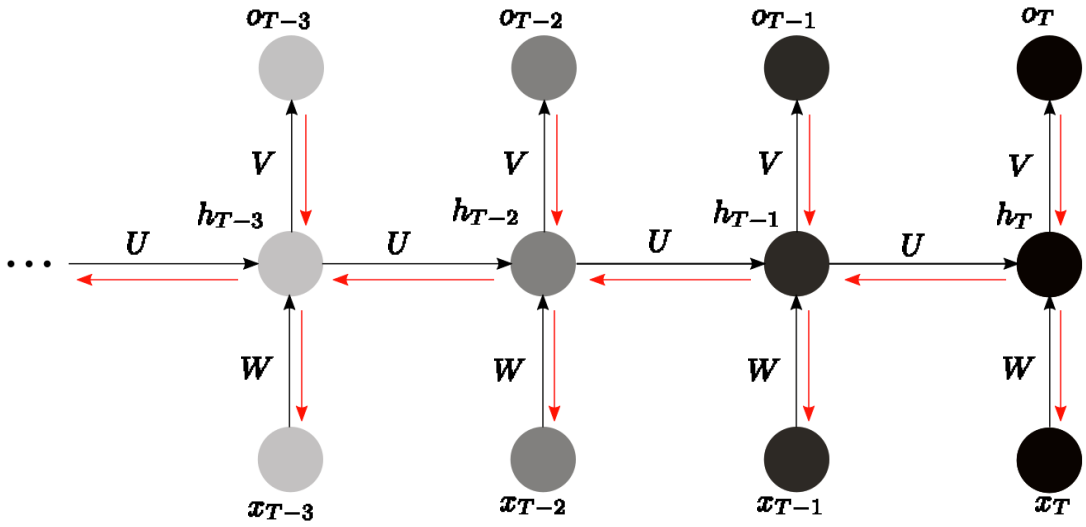


Figure 20: Unfolded RNN forward pass and error propagation

This form is known as a many-to-many RNN, so called because we have multiple inputs and multiple outputs. There are other forms as well, though the concepts of forward and backpropagation remain largely the same.

$$x_i \in \mathbb{R}^{4\times 1}$$

$$W \in \mathbb{R}^{100\times 4}$$

$$U \in \mathbb{R}^{100\times 100}$$

$$h_i \in \mathbb{R}^{100\times 1}$$

$$V \in \mathbb{R}^{2\times 100}$$

$$o_i \in \mathbb{R}^{2\times 1}$$

During forward propagation, we multiply the inputs by the relative weights, and pass the resulting multiplication through any activation functions. As an example, we give the forward propagation in the very last layer:

$$o_T = \text{softmax}(Vh_T)$$

$$h_T = \tanh(Uh_{T-1} + Wx_T)$$

Recursively, we can calculate all of the hidden units and the matrix multiplications that they consist of till the very beginning of the sequence. Note that the same U, V and W are being used during each timestep. This greatly reduces the number of parameters in the model, however it also means we cannot parallelize the training process as we can with CNN models.

To calculate weight updates, we can backpropagate. We define our loss function to be cross entropy loss. This loss gives us the notion of how close our estimated distribution and true distribution is. In the case of only two variables like in seizures, it is known as binary cross entropy. Suppose we have a model that

predicts for a seizure or false positive for a specific timestep t and gives an output (o_{t1}, o_{t2}), where o_{t1} is the probability of a seizure and o_{t2} is the probability of a false positive. Let's say for the same timestep we have a ground truth value of (y_{t1}, y_{t2}), where either y_{t1} or y_{t2} will be exclusively 1. The cross entropy at that point is

$$\mathcal{L}(y_t, o_t) = -\sum_i y_{ti} \log o_{ti}$$

Note a ground truth will cause one of the y_{ti} values to be 1 while the others are 0. Also note that if the probabilities $o_{ti} = y_{ti}\ \forall i$, then $\mathcal{L}(y_t, o_t) = H(y_t)$, because cross-entropy can be thought of as $\mathcal{L}(y_t, o_t) = H(y_t) + D_{KL}(y_t || o_t)$

Proof:

$$-\sum_i y_{ti} \log o_{ti} - H(y_t)$$

$$= -\sum_i y_{ti} \log o_{ti} + \sum_i y_{ti} \log y_{ti}$$

$$= \sum_i -y_{ti} \log o_{ti} + y_{ti} \log y_{ti}$$

$$= \sum_i y_{ti} (\log y_{ti} - \log o_{ti})$$

$$= \sum_i y_{ti} \log \frac{y_{ti}}{o_{ti}}$$

$$= D_{KL}(y_t || o_t)$$

From the inclusion of KL-Divergence, it is readily apparent how cross entropy measures the similarity of distributions true and estimated distributions. In the case of our RNN, note we have multiple outputs. In

this case, we will treat the whole sequence as one training example, and simply add up the cross-entropy error at each output.

$$\mathcal{L}(y, o) = -\sum_t \sum_i y_{ti} \log o_{ti}$$

In the cases where we have a ground truth, this will simplify to

$$\mathcal{L}(y, o) = -\sum_t y_t \log o_t$$

where y_t is the ground truth class at each timestep, while o_t is the corresponding predicted probability for that class.

Our goal is to calculate updates for U, V and W. Since we are summing up all of the errors, it follows that the gradient update will be the sum of the gradients. i.e. if $\mathcal{L} = \mathcal{L}_T + \mathcal{L}_{T-1} + \cdots + \mathcal{L}_0$, then $\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}_T}{\partial W} + \frac{\partial \mathcal{L}_{T-1}}{\partial W} + \cdots + \frac{\partial \mathcal{L}_0}{\partial W} = \sum_t \frac{\partial \mathcal{L}_t}{\partial W}$. We use the chain rule to calculate the gradients for the various matrices at a specific time point t.

$$\frac{\partial \mathcal{L}_t}{\partial V} = \frac{\partial \mathcal{L}_t}{\partial o_t} \frac{\partial o_t}{\partial V}$$

$$\frac{\partial \mathcal{L}_t}{\partial U} = \frac{\partial \mathcal{L}_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \sum_{k=0}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial U}$$

$$\frac{\partial \mathcal{L}_t}{\partial W} = \frac{\partial \mathcal{L}_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \sum_{k=0}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Consider here how the weight matrix V only depends on a single state, while U and W will both depend on inputs from previous instances of themselves, leading to the recurrent use of the chain rule. Similar to the loss, we then take a sum over all time-points to calculate the total gradient.

$$\frac{\partial \mathcal{L}}{\partial V} = \sum_{T} \frac{\partial \mathcal{L}_t}{\partial V}$$

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{T} \frac{\partial \mathcal{L}_t}{\partial U}$$

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{T} \frac{\partial \mathcal{L}_t}{\partial W}$$

These gradients would then be used to update the weight matrices using some update rule, i.e. $U = U - \eta \frac{\partial \mathcal{L}}{\partial U}$, where η is the learning rate. While RNNs have memory, it is finite, with earlier layers contributing less than later layers due to the vanishing gradient effect.

Further considering $\frac{\partial h_t}{\partial h_k}$, we note that it will also have to be computed using the chain rule, as each h_t is only a function of h_{t-1}. This will give long sequences of chained derivates, especially when calculating for the initial layers. From [71] we see that the Jacobians of the $\tanh(\cdot)$ and $\mathrm{sigmoid}(\cdot)$ are upper bounded by 1 and $\frac{1}{4}$ respectively. This means as our chains of derivative multiplications get longer, the impact of that layer on the total gradient tends to 0. Depending on what activation functions we use, we can also have gradients that are consistently greater than 1. This will result in gradients tending to ∞, also resulting in a suboptimal learning strategy.

Long Short Term Memory (LSTM) Networks

To propagate a constant gradient over longer time periods, we introduce a more complex structure to the RNN called LSTM networks. The main advancements of LSTM networks is the addition of learnable gating mechanisms that allow the network to learn long term dependencies within the data, while the overall structure of the network remains the same. They were originally introduced by Hochreiter and Schmidhuber [72], and have been used successfully in a variety of fields, especially NLP [73]. LSTMs introduce an intermediary memory cell that also has a recurrent connection that allows for the error to be propagated without being diminished, addressing the vanishing gradient problem of RNNs.

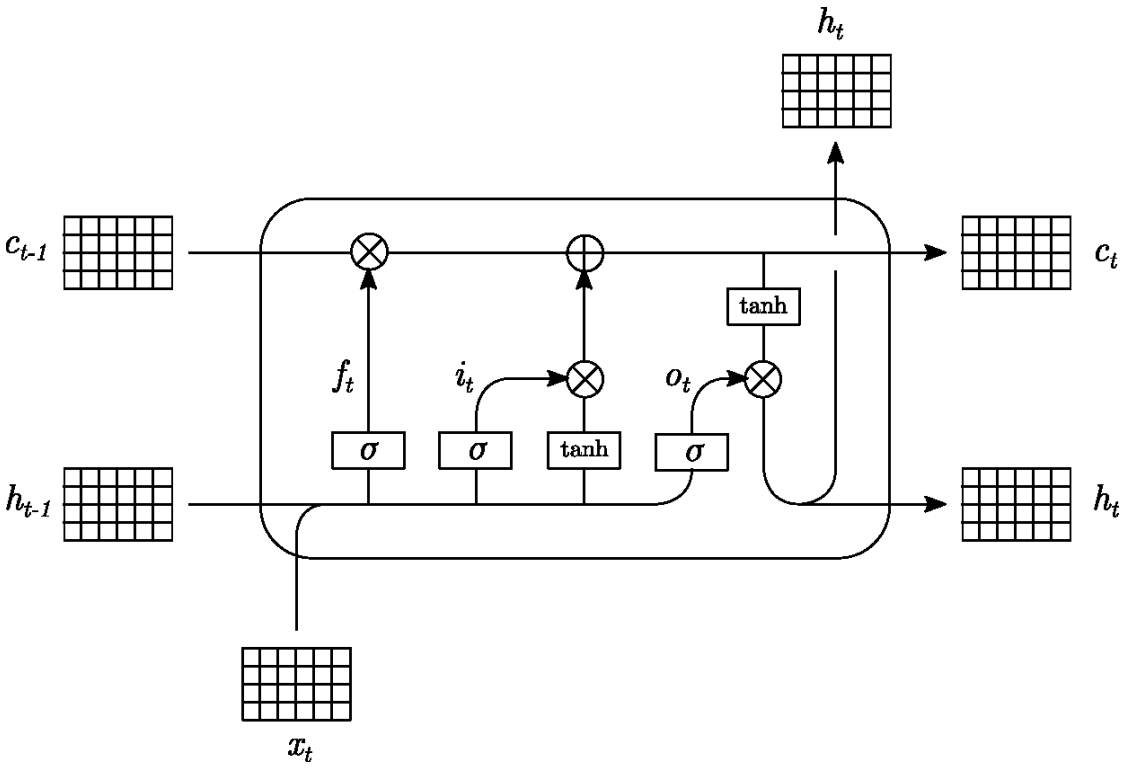


Figure 21: LSTM block internals

The sizes of the different components are provided. B is batch size, D is dimensionality, and H is the number of hidden units of the LSTM cell. In our use case, $B = 100$, $D = 4$, and $H = 100$ in the case of a vanilla LSTM.

$$x_t \in \mathbb{R}^{B \times D}$$

$$f_t \in \mathbb{R}^{B \times H}$$

$$i_t \in \mathbb{R}^{H \times B}$$

$$o_t \in \mathbb{R}^{H \times B}$$

$$h_t \in \mathbb{R}^{B \times D}$$

$$c_t \in \mathbb{R}^{H \times B}$$

The three gates of an LSTM are:

Forget gate: Included in 'vanilla' LSTMs nowadays, they were originally introduced in 2000 by Gers et. al., to prevent unbounded growth of the cell state. The forget gate is there to be selective about what information we should remember by looking at a concatenation of the previous hidden state h_{t-1} and the current input x_t. After being passed through a learnable weight matrix W_f, the sigmoid function will output a value between 0 and 1 which will perform a Hadamard product (pointwise multiplication) with the cell state. $W_f \in \mathbb{R}^{H \times 2B}$

$$f_t = \sigma\big(W_f[h_{t-1}, x_t]\big)$$

Input: We also allow the network to selectively add new information into our cell state. The sigmoid layer once again decides which values to update, but then this is directly pointwise multiplied with a new set of candidate cell states $\widetilde{c_t}$ which are made by passing the concatenated input through a $\tanh(\cdot)$ layer

rather than a sigmoid layer to force that values to be between -1 and 1. The result is added onto the currently existing cell state.

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$
$$\widetilde{c_t} = \tanh(W_c[h_{t-1}, x_t])$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

Output: Finally we permit the network to selectively decide what information to output. This gives the network freedom to create dependencies between both long-term dependencies stored in the cell state, and recent, possible ephemeral information. The sigmoid is applied on a standardized version of the cell-state which is passed through a $\tanh(\cdot)$ function to compress it between -1 and 1. The output will also be the new hidden state of the LSTM cell.

$$o_t = \sigma(W_o[h_{t-1}, x_t])$$
$$h_t = o_t \circ \tanh(c_t)$$

While LSTMs effectively handle the gradient vanishing problem, they are still susceptible to exploding gradients, and as such it is common to use gradient clipping when experimenting with different LSTM architectures. There are also several different variants on this LSTM architecture such as the addition of peephole connections [74] and the GRU [75].

CNN-LSTM

While LSTMs will generally provide good performance, they were difficult to train and expensive in terms of training time due to lack of parallelization. Instead, we decided to add a deep convolutional network between the input data and LSTM for feature extraction. A CNN layer consists of performing a

convolution across the input space using randomly initialized kernel functions. The completed convolution will be referred to as a feature map, and each layer can have a user-defined number of feature maps of the data. As each feature map is computed on the same data, CNNs are highly efficient algorithms when parallelized. A simple example of a convolution with kernel [1, 1, 1] is given below. During implementation, all of these kernels will be initialized randomly. Note as we are working with a discrete input, the convolution is just a cumulative sum of the point wise multiplication for each convolutional window.

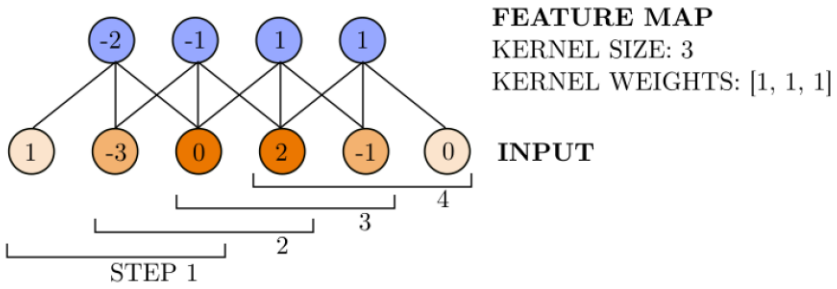


Figure 22: Convolutional Layer toy example

Convolutional networks are mostly used in image-based problems. In our case, we do not necessarily want convolutions between our features, but rather just across time. To implement this we use 1D convolutions. Consider our input dataset $\boldsymbol{X} \in \mathbb{R}^{4500\times4}$. We can take a kernel size $(k \times 4)$, and then slide that kernel across. This is a 1D convolution. In future iterations of the algorithm, it may make more sense to perform 2D convolutions to find features dependencies. Some seizure subtypes have subtle absolute feature data, but strong correlations between heart rate and accelerometer movement.

After each convolutional layer, we often apply pooling layers. Max pooling, taking the maximum of n neighboring values in the feature map over the entire space, is the most common. It provides a regularization effect, transformation, rotation and scaling invariance, as well as dimension reduction.

Additionally, it has become common practice to apply batch normalization layers after each hidden layer to counteract Internal Covariate Shift (ICS). Learning theory is based on the assumption that all data (both training and testing) and independent and identically distributed (i.i.d.). We whiten our data before input, but as we propagate through the layers, the distribution of the features (covariates) will slowly begin to drift. This is known as ICS. The later layers will subsequently need to adapt to this drift, significantly slowing down learning [76].

The idea of batch normalization is to restrict the activations of each layer to be to be standardized with 0 mean and unit variance. The theory is that this will whiten the distribution after each layer, accelerating network training. In practice, such a strict restriction would hinder the expressive power of the network, so we add in learnable parameters γ and β allowing the network some freedom in whitening the data.

Taking $\boldsymbol{X} = \{x_1, \dots, x_n\}$ to be one batch in our dataset, and $\epsilon \sim 0$ for numerical stability,

$$\mu \leftarrow \frac{1}{m}\sum_i x_i$$

$$\sigma^2 \leftarrow \frac{1}{m}\sum_i (x_i - \mu)^2$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\mathrm{BN}_{\gamma,\beta} = \gamma \hat{x}_i + \beta$$

We also point out that while the ICS has been the commonly accepted reason behind batch normalization, recent work from Santurkar et. al. [77] suggests this is not the case. They conclude that while batch normalization does whiten layer outputs, its fundamental effect is to make a smoother optimization landscape, inducing more stable gradient behavior, and resulting in faster training.

Finally to further regularize our model, we also consider using dropout layers. While expressive systems allow us to generate complex decision functions, they will also have a tendency to overfit. Similar to bagging in decision trees, we can reduce variance by taking the average over a large number of networks. Unfortunately training this many networks is computationally infeasible. Instead we randomly drop out network neurons and their connections during training of each batch. This will prevent co-adaptation of neurons over time. When testing a new input, we take the average of all the neurons, multiplied by $(1 - p)$, where p is the probability of randomly dropping a neuron. In practice, this results in performance similar to averaging a large batch of networks [78]. Dropout regularization would be applied on top of the regularizing effects of both max-pooling and batch normalization.

TCN

Despite the efficacy of recurrent models in sequence modeling tasks, recent research has shown that CNNs can also achieve state-of-the-art accuracies on specific tasks. This raises the question as to whether Convolutional networks are successful due to a specific domain application, or because they can inherently be used in general sequence modeling applications.

Let's explore this problem with at a toy example. Suppose we are given an input sequence $(x_0, \ldots, x^{\top})$ and wish to predict a corresponding output $(y_0, \ldots, y^{\top})$ for each time point. In RNNs, there is a causal framework. All information to predict an output at a specific timepoint only uses information that has come previously. Thus to calculate y_t, we must only use $x_0, \ldots x^{\top}$. Additionally, since our goal is sequence modeling, note the second constraint that the output size must match that of the input.

Generally CNNs will break both these constraints. Convolution kernels use data from the past and the future, and each convolution will shrink the dataset in the time dimension. To ensure causality, we define

a causal convolution [79], as a convolution where all inputs to kernels are coming strictly from the past. This architecture is similar to that of the time-delay network. To maintain the shape of the output, we use zero-padding on one side.

A RNN-like CNN can now be defined. Unfortunately, covering long sequences with regular convolutions will result in a very deep architecture consisting of large features. In real-time applications, the network would have to either be optimized, distilled or otherwise compressed to allow for fast calculations.

Dilated Convolutions

Following the work of Wavenet [80], and Yu and Koltun [81] the original TCN authors [79] employed dilated convolutions to exponentially increase the receptive field of neurons in later layers. We define dilated convolutions as

$$F(s) = (\boldsymbol{x} *_d f)(s) = \sum_{i=1}^{k} f(i) \cdot \boldsymbol{x}_{s-d \cdot i}$$

where $\boldsymbol{x} \in \mathbb{R}^N$ is our 1 dimensional input sequence, k determines the kernel size, and $F(s)$ signifies the s^{th} element of our feature map F. Note the $s - d \cdot i$ term that dilates the convolution. When $d = 1$, a dilated convolution is reduced to a normal convolution. An example of dilated convolutions can be seen in Figure 23. Note how as we increase the dilation at each layer, the receptive field gets exponentially larger. In the example, the receptive field of a neuron in the output layer is 16. In comparison if we had used a regular convolutional network with the same strides and kernel sizes, the receptive field would be 5. We can increase or decrease the size of the receptive field by varying the filter size and the dilation rate.

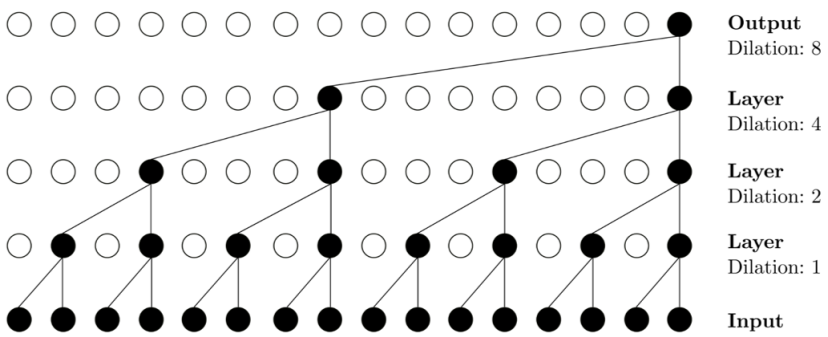


Figure 23: Dilated causal convolutions

To calculate the receptive field of the TCN, we look at the kernel size and dilation parameters at each layer, and take a recursive sum. This formulation works under the assumption that each element of a kernel will have receptive fields that either touch or overlap on the layer below. If they do not, there will be gaps in the lower layers. Layer 0 will always have a dilation of 1 and kernel size of 1, as it is simply the input sequence. Defining $R_F(l)$ as the receptive field at layer l,

$$R_F(0) = 1$$

At each subsequent layer, the receptive field will be calculated by

$$R_F(l) = [K(l) - 1]d(l) + R_F(l-1)$$

Note that in the setup of a standard TCN, each residual block will have the same convolutional layers occurring twice.

Residual Block

During training of deep networks, there is empirical evidence that after a certain depth, network performance saturates, and counterintuitively begins dropping [82]. Even more surprisingly, this performance degradation is not due to model overfitting, as [82]–[84] all show an increase in training error when performance begins to degrade. Theoretically, this should not happen any subnetwork placed on top of some optimal shallow network has the expressive capabilities to learn an identity mapping. At worst, deeper layers should maintain the performance of shallower layers.

A novel method called Residual learning [82], addresses this problem with the introduction of residual skip connections. Instead of trying to make the higher subnetwork learn a residual mapping, we provide it with an identity mapping, and allow it to create a residual mapping if it needs to add any extra information. Mathematically, if the output of the optimal shallow network is x, and the output of the higher subnetwork is $F(x)$, we add a connection directly routing x across $F(x)$. The cumulative output that is then propagated is $H(x) = F(x) + x$. This is shown in Figure 24.

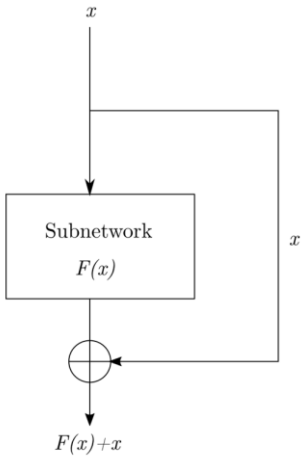


Figure 24: Residual Connection

Should the identity mapping be ideal for performance, it is easier for the network to optimize weights of the subnetwork $F(x)$ to be 0, rather than trying to learn the identity mapping via backpropagation through many non-linear layer stacks. As TCN's will generally be deep networks to learn long sequences, it is important that these residual connections are present. This is seen going through the 1x1 convolution in Figure 25. We use the 1x1 convolution to downsample the input should the need arise, so that summation with the output is possible.

To counter the problem of vanishing/exploding gradient, weight normalization is used after every dilated convolution layer. Weight normalization [85] is similar to batch normalization in that it normalizes layer weights. Instead of normalizing the mean and standard deviation, weight normalization normalizes based on each vector's orientation and magnitude, essentially separating the norm from the direction. It is calculated by reparametrizing the weight of each weight vector as

$$\boldsymbol{w} = \frac{g}{||\boldsymbol{v}||}\boldsymbol{v}$$

g and $\boldsymbol{v}$ are then optimized through gradient descent. Weight normalization is more deterministic than batch normalization, and is computationally simpler. From the original weight normalization paper, we see that weight normalization is faster than batch normalization, though each performs better in specific situations. Finally a dropout layer is also added to provide regularization to the network, leading to the final residual block shown in Figure 25, as described in [79].

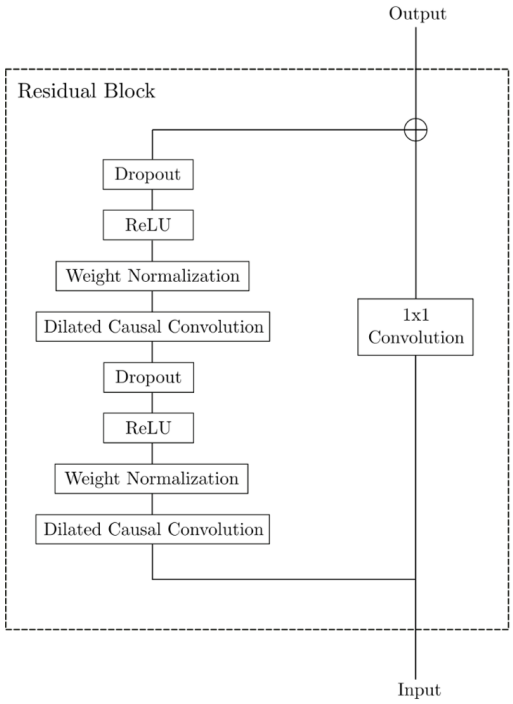


Figure 25: Residual Block of TCN

These residual blocks, when stacked on top of each other, are known as a Temporal Convolutional Network.

Network Architectures and training

LSTM

We used a stacked LSTM model with dropout regularization (dropout rate: 0.2) applied on dense-layers between the LSTM. We do not add dropout inside the LSTM blocks as if could encourage randomly forgetting some long-term dependency. Each LSTM cell has a70 hidden units, culminating with a final SoftMax layer for a probabilistic output decision. The learning rate was 0.02, with a categorical cross-entropy loss function for backpropagation. Batch size was 100, and training accuracy converged to approximately 88% within 10 epochs. The batch size for each update step was 100.

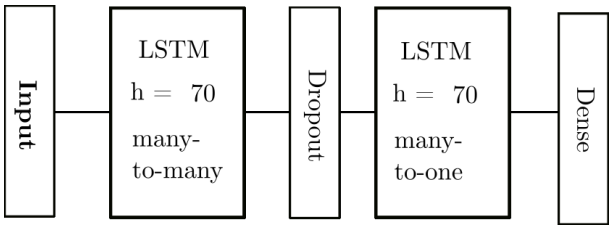


Figure 26: LSTM architecture

CNN-LSTM

To speed up LSTM training we decided to use a CNN as a feature extraction step. By design, CNN computations can occur in parallel (same filter applied to multiple locations of the image at the same time), leading to large processing time gains. The CNN will additionally serve as a feature extraction mechanism which can be passed on to classical classification algorithms. The architecture is three 1D CNNs with padding to maintain shape, batch normalization (on the feature axis), 1D max pooling (pool size: 2) and dropout (dropout rate: 0.1). Two LSTMs (70 hidden units) were stacked onto the final CNN layer, culminating with a softmax output. Batch size was 100, and validation accuracy converged to 93% after 10 epochs. The batch size for each update step was 100.

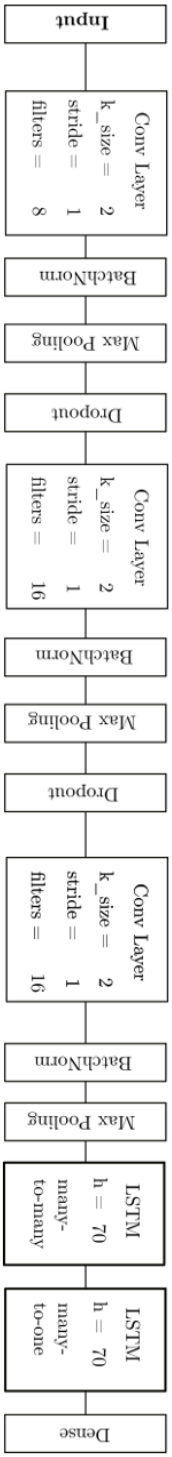


Figure 27: CNN-LSTM architecture

Table 17: CNN-LSTM architecture summary

Layer	Kernel Size	Stride	Filters	Dropout (%)	Hidden Units	Output Shape
Input	-	-	-	-	-	4500 x 4
Conv 1D	2	1	8	-	-	4500 x 8
Feature BN	-	-	-	-	-	4500 x 8
Max Pool 1D	2	2	-	-	-	2250 x 8
Dropout	-	-	-	0.1	-	2250 x 8
Conv 1D	2	1	16	-	-	2250 x 16
Feature BN	-	-	-	-	-	2250 x 16
Max Pool 1D	2	2	-	-	-	1125 x 16
Dropout	-	-	-	0.1	-	1125 x 16
Conv 1D	2	1	16	-	-	1125 x 16
Feature BN	-	-	-	-	-	1125 x 16
Max Pool 1D	2	2	-	-	-	562 x 16
LSTM 1	-	-	-	-	70	562 x 70
LSTM 2	-	-	-	-	70	70
Dense	-	-	-	-	-	2

TCN

We implemented a generic TCN in Keras following [79]. As weighted convolutions have not yet been implemented in Keras, we have bypassed that layer in the residual blocks. We have also implemented skip connections to add the outputs of every residual block to the final output. Skip connections can alleviate the vanishing gradient problem, and enhance feature propagation in deep networks. They have commonly been used in networks like DenseNets. In addition to these skip connections, the residual blocks will still contain their own identity mapping functions.

We selected dilation values of [1, 2, 4, 8, 16, 32, 64]. We select stacks of 1 residual blocks for each dilation value, with a dropout value of 0.05. All convolutional layer had 20 filters with a kernel size of 20. Together this setup led to each kernel element on the top layer to have a receptive field of 4827, covering our entire sequence. The output layer was softmax for a probabilistic decision. Each model was trained for 10 epochs within which validation accuracy converged to over 99% with the ADAM optimizer. The batch size for each update step was 100.

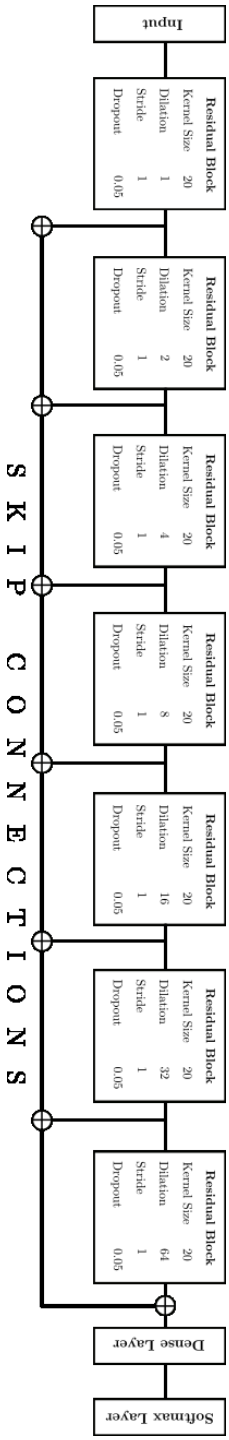


Figure 28: TCN architecture

Results

Evaluation Metrics

ROC and precision-recall curves were used to evaluate the detector performance. We motivate the use of a precision-recall curve as there will be a dataset imbalance during testing (48 False Positive Segments to 4 seizure segments during each fold). The precision metric measures the posterior probability of a segment being a seizure, given the detector saying it was. It answers the question, *"how many detections are relevant?"*. Recall is a synonym for sensitivity, and answers the question *"How many relevant segments are detected?"*.

$$\text{Precision} = P\left(Y = 1 \middle| \hat{Y} = 1\right) = \frac{TP}{TP + FP}$$

$$\text{Recall} = \text{Sensitivity} = P\left(\hat{Y} = 1 \middle| Y = 1\right) = \frac{TP}{TP + FN}$$

$$\text{False Positive Rate} = P\left(\hat{Y} = 1 \middle| Y = 0\right) = \frac{FP}{FP + TN}$$

Note as this is a second stage detector, there was no need to calculate the false positive rate in a unit time. We setup false positive rate as a percentage, and used in-vivo results from the isolation forest detector to estimate the amount of false positives per unit time.

To evaluate the ideal threshold, we used a normal and weighted F_1-score, a metric that scores the detector at specific thresholds according to the harmonic mean of its precision and recall. The harmonic mean is defined as the reciprocal of the arithmetic mean of the reciprocals of a given set of observations, and is one of the three classical Pythagorean means. It is more natural to use harmonic mean here as precision and recall share the same numerator. This leads to a higher punishment of extreme values. As a simple

example, if precision is 0 and recall is 1, the arithmetic mean would give 0.5, while the harmonic mean would give 0.

$$F_1 = \left(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Note in the traditional F_1 score, both precision and recall are weighed equally. In general, we want to place a higher emphasis on recall, due to the consequences of missing a seizure. This can be done by using the β-weighted F score.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

$\beta > 1$ will weigh recall higher than precision, while $\beta < 1$ will weigh precision higher than recall. In our experiments we use $\beta = 1.5$. For all our experiments, we shall use either F_1 or F_β to set the threshold value for our detector.

Offline Cross Validation

4-fold cross validation was performed on the dataset. During each fold, 48 windowed false positive segments, and 4 windowed seizure segments were randomly sampled without replacement for testing. The 144 remaining false positive segments and 18 seizure segments were used for testing. Note that there are 2 randomly chosen seizures that will never be tested during cross-validation. We sample without replacement so all seizures and false positives have a chance to be tested as an unseen example.

Accumulation Filter

A 1st order IIR filter is used to soften the output of the neural networks. This filter is identical to that used in the anomaly detection stage of the algorithm.

$$y(t) = \alpha x(t) + (1 - \alpha) y(t - 1)$$

As high detector outputs on false positive data is expected to be sporadic, the filter parameter was set to $\alpha = 0.05$. This heavy weighting towards the previous value will create a very slow filter, as seen in Figure 29. As seizures should have a near continual segment of high network outputs, a slow filter allows us to set a threshold between the seizures and the false positives. If the filter is too fast, any short burst of seizure like activity will cause the filtered output to hit the threshold.

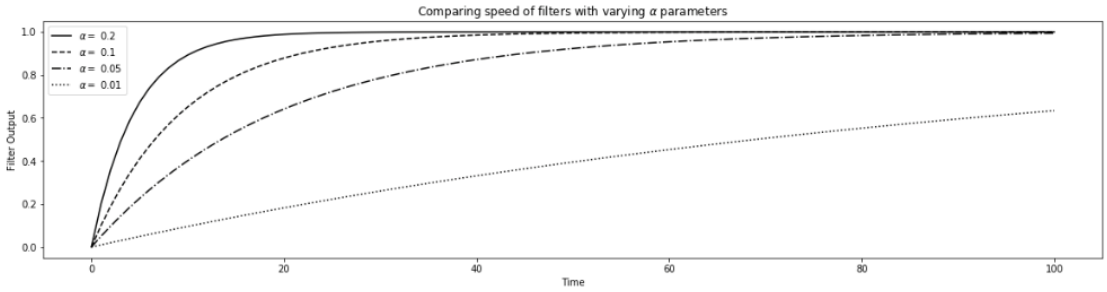


Figure 29: Accumulation Filter comparisons

We will not do grid search to find the optimal value of the filter parameter. During cross validation, the maximum attained value of the accumulation filter for each test segment is retained, and we can describe our ROC and Precision Recall curves by varying the threshold across these maximum values. Note the baseline curves in both ROC and Precision Recall graphs. These lines represent a baseline hypothesis where you simply guess whether a specific event is a seizure or a false positive.

Each CV-ROC curve will also contain curves from every fold of the cross validation, as well as a standard deviation region. There will be two mean ROC curves plotted. One will be the curve made from all the data plotted on one chart. The other is a mean of the curves from each fold.

LSTM

The validation accuracy for LSTM converged to 89% within 10 epochs. Without GPU access, training time was approximately 20 hours per fold on a CPU.

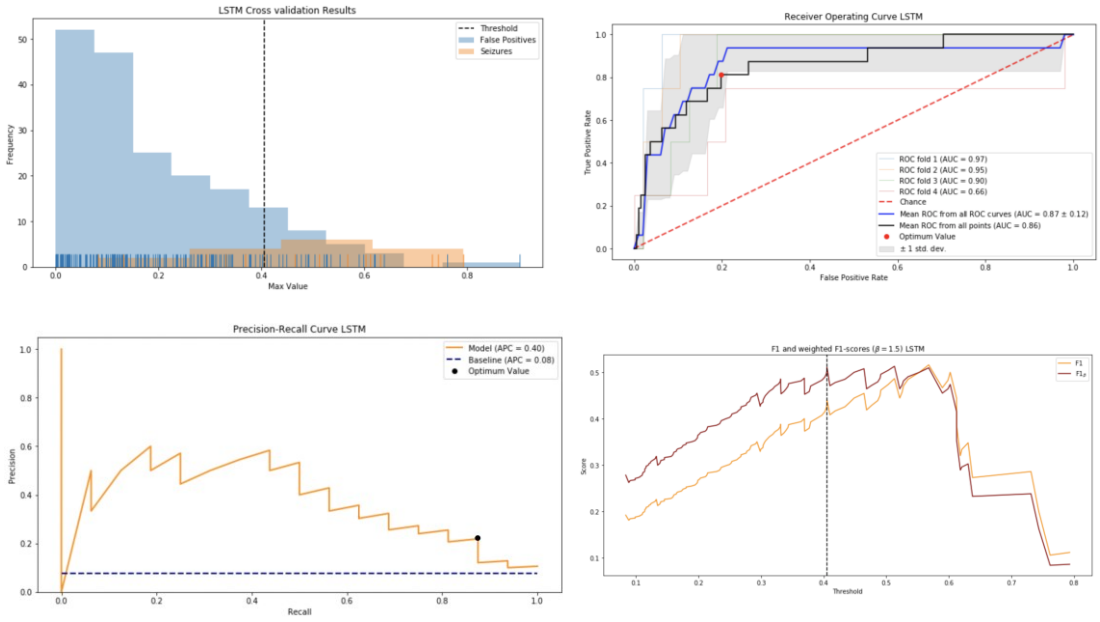


Figure 30: Cross validation results for LSTM network. a) shows a rug-plot and histogram of maximum values reached by the accumulation filter for seizures and false positives. b) shows the corresponding ROC curve. c) shows the precision-recall curve. d) shows the F1 and weighted F1 scores for varying thresholds.

Table 18: Optimal LSTM characteristics

Optimal Threshold	0.4053
Sensitivity	81.25%
Specificity	79.10%

LSTM results were encouraging as a baseline for this task. At an optimal configuration, a sensitivity of approximately 80% is not acceptable in our application. If we lower the threshold to allow for 90% accuracy, the false positive rate will be around 0.7, which translates to an estimated 0.903/day using our in-vivo isolation forest results. With the additional increase in latency, it would be inefficient to implement a secondary detection stage with these characteristics. We note the strict performance penalties of the precision-recall curve on anomaly detection activities. Despite a good overall false positive rate, the lack of true positives drive down the precision metric to 0.3. Since we require a idealized characteristic of 1 false positive per seizure, an acceptable algorithm will require a precision of at least 0.5.

CNN-LSTM

The validation accuracy for CNN-LSTM converged to approximately 97% accuracy within 10 epochs with a training time under 30 minutes per epoch on a CPU.

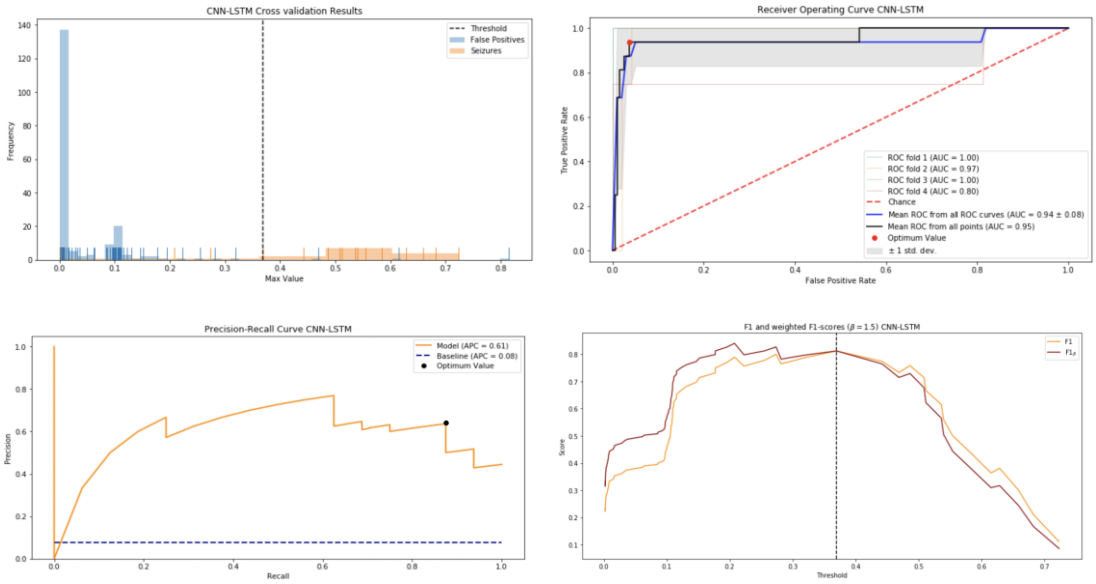


Figure 31: Cross validation results for CNN-LSTM network. a) shows a rug-plot and histogram of maximum values reached by the accumulation filter for seizures and false positives. b) shows the corresponding ROC curve. c) shows the precision-recall curve. d) shows the F1 and weighted F1 scores for varying thresholds.

Table 19: Optimal CNN-LSTM characteristics

Optimal Threshold	0.3696
Sensitivity	93.70%
Specificity	96.42%

With this specificity, we can expect a mean False Positive Rate of 0.04644/day. A sensitivity of over 93% is an acceptable characteristic. We note that the two seizures that were not detected during cross validation still reached maximum values of 0.22 and 0.28 respectively. Despite being atypical seizure sements, using old sensor data with gaps, we think it likely that careful hyperparameter tuning and/or a deeper network may allow for a higher AUC. It is also likely that with the newer watches, our sensitivity will be higher than predicted here. From the precision recall curve, we see an optimal precision value above 0.5, meeting our idealized expectation of a 1:1 seizure to false positive ratio. There are still three false positives that are confidently identified as seizures. We cannot validate what activities they are as old Video EEG records are not stored in our system, but from a spectrogram we see extended bands of activity that mimic periodic activity (running, brushing teeth, weightlifting). Additionally, heart rate increase was marginal, with none of the 3 crossing 100 bpm. This suggests we may have to find a method of increasing the weighting of heart rate features in the network.

TCN

The validation accuracy for TCN converged to 99% within 10 epochs. Training time was approximately 1 hour per epoch on a CPU.

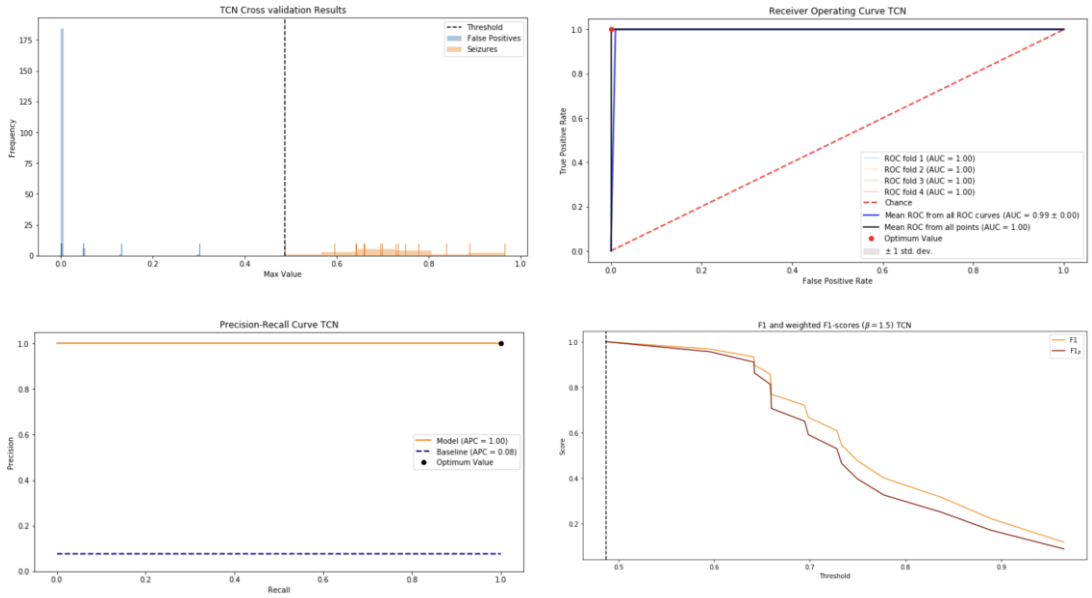


Figure 32: Cross validation results for TCN network. a) shows a rug-plot and histogram of maximum values reached by the accumulation filter for seizures and false positives. b) shows the corresponding ROC curve. c) shows the precision-recall curve. d) shows the F1 and weighted F1 scores for varying thresholds.

Table 20: Optimal TCN characteristics

Optimal Threshold	0.48
Sensitivity	100%
Specificity	100%

TCN shows perfect separation of seizures and false positives over 5-fold cross validation. To ensure overfitting was not occurring, numerous cross-validations were performed to verify these results. Three false positives are noticeable separate from the majority that are clustered around 0. The same false positives can be seen in the CNN-LSTM model across the threshold line. This suggests that the model is indeed learning about some inherent structure that allows for good separation. Additionally, we see that on average, maximum thresholds of seizure segments are higher on average (0.72) as compared to LSTM

(0.49) and CNN-LSTM (0.57). The model is considerably more sure that sections are seizures, as the confidence of output needs to be consecutively ~1 for almost 30 seconds to reach this value with our accumulation filter. This algorithm will significantly improve latency if implemented.

Summary

Table 21: Summary of performance characteristics for 2nd stage detector

	Sensitivity	**Specificity**	**AUC**	**APC**
LSTM	81.25%	79.10%	0.87	0.4
CNN-LSTM	93%	96%	0.94	0.61
TCN	**100%**	**100%**	**1**	**1**

Real Time Detection

Watch Implementation

Despite the impressive results from the TCN model, the dilated causal convolution layers were difficult to implement efficiently in Swift. There is a conversion tool available called CoreML assists in converting some common layers into a C++ wrapper around the low-level Metal Performance Shaders. The conversion allows for optimized processing of the data through the network, and uses the GPU on the target device if available. With some configuration, were able to convert CNN and LSTM layers from Keras to CoreML layers. For now we were not able to convert layers in PyTorch or Tensorflow as cycles have not yet been implemented.

All 22 seizures and 192 false positives were windowed and used to train a final CNN-LSTM model in Keras. The weights and biases of all the layers were frozen, and then all the layers were converted into CoreML compatible layers and reassembled. The resulting model was imported into Swift. It expected an input vector of the shape (4500,4), and will give a corresponding probabilistic output of likelihood that the input segment is a seizure or False Positive. Together with the Isolation Forest detector, we can summarize the full detection algorithm.

Table 22: Full detection algorithm

1. Buffers collect accelerometer and heart rate data constantly
2. Every 5 seconds, 5 second buffer chunks are passed through the anomaly detector class, which filters the segments and extracts selected features
3. Each 1 second chunk is passed to the isolation forest which calculates path lengths decides whether to classify it as an anomaly or not
4. All outputs are passed through an accumulation filter that provides robustness against spurious detections.
5. If accumulation filter passes a threshold of 1.55, the isolation forest processing pipeline shuts off.
6. A running buffer of 45 seconds is interpolated, filtered and standardized, then passed through the CNN-LSTM which outputs a probability of the segment being a seizure.
7. This output is passed through a secondary accumulation filter. If this filter passes a threshold of 0.40, the seizure detection protocol will be triggered.

Note are that we set the threshold to 0.40 as it does not change the sensitivity of the detector but will make separating false positives easier. This can be adjusted to improve latency and sensitivity if required.

In-Vivo Results

Across all 30 patients, we tracked for a total of 2004 hours, giving an average of 65 hours per patient. We kept a track of the number of Isolation Forest detections, and the number CNN-LSTM detections that occurred, to evaluate performance against the initial stage. We also validated every detection with video EEG for standard reference. Latency was calculated from the beginning of convulsions.

Table 23: In-vivo statistics for 2nd stage detector

Total Hours Tracked	Average Hours per patient	False Positives	True Positives	Total Forest Detections	Mean Latency ± std (s)
2004	65	4	12	109	62 ± 10

Table 24: Raw in-vivo statistics for 2nd stage detector

		Detector		Total
		(+)	(-)	
Video EEG	(+)	12	0	12
	(-)	4	109	97
Total		16	93	109

Taking the Video EEG as a reference standard, we see a sensitivity of **100%** over 12 seizures, and a specificity of **96.4%** over 113 false positive detections from the isolation forest, corresponding to a false positive rate of **0.05 /day**. This is in agreement with our expected false positive rate of **0.04644 /day** estimated during cross validation. Our 95% Clopper-Pearson confidence intervals were **(73.5%, 100%)** for sensitivity and **(91.1%, 99.0%)** for specificity. The sensitivity interval is so wide because we only recorded 12 seizures. Latency is also on the threshold of being acceptable. It is noted that this latency is the same latency predicted in offline analysis of the isolation forest (mean 62 seconds), suggesting either the CNN-LSTM is classifying very quickly, or lack of accelerometer data gaps in the newer generation watches have caused quicker isolation forest detection. Additionally, it is noted that many seizures had heart rate gaps during the tonic phase. For seizures with no heart rate gaps, latency was averaged at 52 seconds. This suggests that data gaps may still be affecting seizure detector performance.

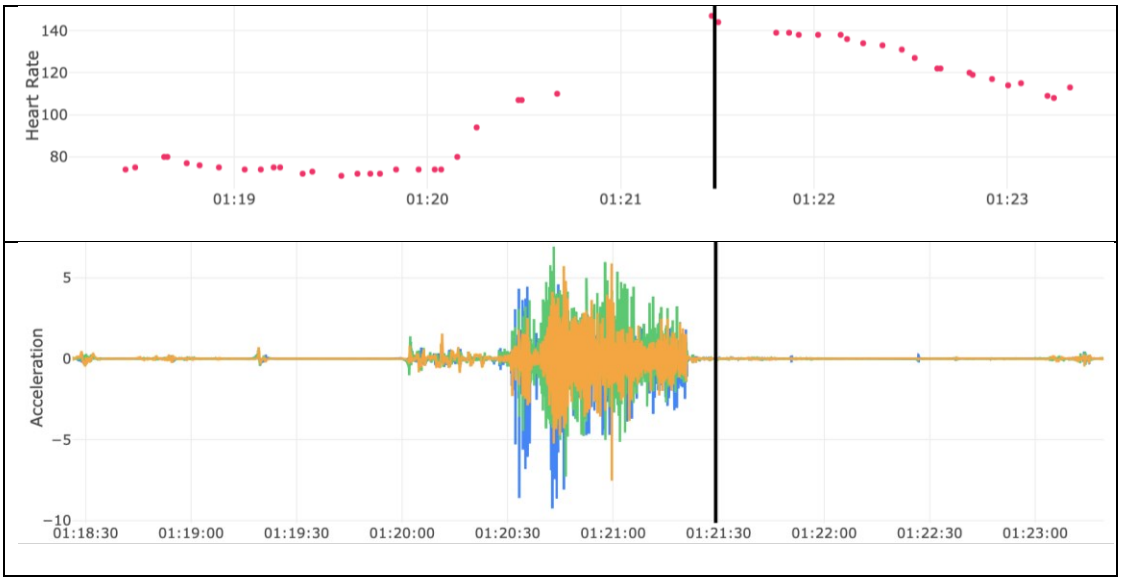


Figure 33: Data gap illustration on seizure data. a) shows the heart rate dat. b) shows corresponding accelerometer data.

Finally, the detector was also used by some Beta users to assess its efficacy in ambulatory individuals. The FPR was far lower than the original anomaly detector, and is quite surprising given no ambulatory data was used to train the CNN-LSTM. This suggests that the false positives from the EMU have characteristics that mimic ambulatory false positives. As before, the false positives were primarily from monotonous physical activity.

Conclusion

We have introduced the first smartwatch-based tonic-clonic seizure detector in this thesis. Our method, an ensemble of isolation forest and CNN-LSTM detects anomalies in real time data, then classifies them as either seizures or false positives with a sensitivity of 100% CI (73.5%, 100%) and a specificity of 96.4% CI (91.1%, 99.0%), corresponding to a false alarm rate of 0.05/24h at a latency of 62 seconds. This false alarm rate is state-of-the-art for commercial seizure detectors, and the latency is just inside the clinical window. We have covered feature extraction and selection using mutual information techniques and theory of various anomaly detection models showing the advantages of the isolation forest model. We have also covered the theory of various deep learning classification models and performed cross-validation showing the efficacy of each model in classifying seizures and false positives. We provide a summary of how the algorithms and preprocessing pipeline were implemented in the Apple Watch, and in-vivo results testing our algorithm in EMU patients. Our results demonstrate quantitatively demonstrate the value of such a device in EMU settings and also the potential in ambulatory patients.

Limitations

Despite producing outstanding cross-validation results, we were not able to implement TCN's on the Apple Watch. This by itself could greatly improve all metrics of the detector. Additionally, the high latency suggests a different approach may still be required for the anomaly detection stage. Seeing the success of incorporating causal temporal data into the classifier, a HMM may be a good baseline to begin with. We can also think of using deep learning for anomaly detection. Methods like AnoGAN and Autoencoders have shown promise. Another method would be to use the loss function of a one-class SVM or SVDD on a neural network. An issue with deep learning be CPU usage, but it would be possible to shorten window length, increase the time between processing windows (i.e. 10 seconds), or distill the

network. Finally, as seizures from the same patient always have extremely similar characteristics transfer learning may be possible for customized detectors as we collect more data.

Bibliography

[1] CDC, “Epilepsy Data and Statistics.” [Online]. Available: https://www.cdc.gov/epilepsy/data/index.html.

[2] C. E. Stafstrom and L. Carmant, “Seizures and Epilepsy: An Overview for Neuroscientists,” *Cold Spring Harb. Perspect. Med.*, vol. 5, no. 6, pp. a022426–a022426, Jun. 2015.

[3] P. Kurle and P. Rutecki, “Seizures and Epilepsy,” in *Neurology Secrets*, Elsevier, 2010, pp. 315–339.

[4] A. T. Berg *et al.*, “Revised terminology and concepts for organization of seizures and epilepsies: Report of the ILAE Commission on Classification and Terminology, 2005-2009,” *Epilepsia*, vol. 51, no. 4, pp. 676–685, Apr. 2010.

[5] R. S. Fisher *et al.*, “Operational classification of seizure types by the International League Against Epilepsy: Position Paper of the ILAE Commission for Classification and Terminology,” *Epilepsia*, vol. 58, no. 4, pp. 522–530, Apr. 2017.

[6] R. A. Machado, “Understanding Hyper Motor Seizures,” *Epilepsy J.*, vol. 2, no. 2, 2016.

[7] A. E. Cavanna and F. Monaco, “Brain mechanisms of altered conscious states during epileptic seizures,” *Nat. Rev. Neurol.*, vol. 5, no. 5, pp. 267–276, May 2009.

[8] A. M. Pack, “SUDEP: What Are the Risk Factors? Do Seizures or Antiepileptic Drugs Contribute to an Increased Risk?: SUDEP and Risk Factors,” *Epilepsy Curr.*, vol. 12, no. 4, pp. 131–132, Jul. 2012.

[9] S. Beniczky, I. Conradsen, O. Henning, M. Fabricius, and P. Wolf, “Automated real-time detection of tonic-clonic seizures using a wearable EMG device,” *Neurology*, vol. 90, no. 5, pp. e428–e434, Jan. 2018.

[10] T. Tomson, R. Surges, R. Delamont, S. Haywood, and D. C. Hesdorffer, “Who to target in sudden unexpected death in epilepsy prevention and how? Risk factors, biomarkers, and intervention study designs,” *Epilepsia*, vol. 57, pp. 4–16, Jan. 2016.

[11] A. Schulze-Bonhage *et al.*, "Views of patients with epilepsy on seizure prediction devices," *Epilepsy Behav.*, vol. 18, no. 4, pp. 388–396, Aug. 2010.

[12] C. Hoppe, M. Feldmann, B. Blachut, R. Surges, C. E. Elger, and C. Helmstaedter, "Novel techniques for automated seizure registration: Patients' wants and needs," *Epilepsy Behav.*, vol. 52, pp. 1–7, Nov. 2015.

[13] A. Van de Vel, K. Smets, K. Wouters, and B. Ceulemans, "Automated non-EEG based seizure detection: Do users have a say?," *Epilepsy Behav.*, vol. 62, pp. 121–128, Sep. 2016.

[14] C. E. Elger and C. Hoppe, "Diagnostic challenges in epilepsy: seizure under-reporting and seizure detection," *Lancet Neurol.*, vol. 17, no. 3, pp. 279–288, Mar. 2018.

[15] W. V. Paesschen, "The future of seizure detection," *Lancet Neurol.*, vol. 17, no. 3, pp. 200–202, Mar. 2018.

[16] S. Beniczky, T. Polster, T. W. Kjaer, and H. Hjalgrim, "Detection of generalized tonic-clonic seizures by a wireless wrist accelerometer: A prospective, multicenter study," *Epilepsia*, vol. 54, no. 4, pp. e58–e61, Apr. 2013.

[17] A. Ulate-Campos, F. Coughlin, M. Gaínza-Lein, I. S. Fernández, P. L. Pearl, and T. Loddenkemper, "Automated seizure detection systems and their effectiveness for each type of seizure," *Seizure*, vol. 40, pp. 88–101, Aug. 2016.

[18] A. L. Patterson *et al.*, "SmartWatch by SmartMonitor: Assessment of Seizure Detection Efficacy for Various Seizure Types in Children, a Large Prospective Single-Center Study," *Pediatr. Neurol.*, vol. 53, no. 4, pp. 309–311, Oct. 2015.

[19] S. Ramgopal *et al.*, "Seizure detection, seizure prediction, and closed-loop warning systems in epilepsy," *Epilepsy Behav.*, vol. 37, pp. 291–307, Aug. 2014.

[20] A. Van de Vel *et al.*, "Non-EEG seizure detection systems and potential SUDEP prevention: State of the art," *Seizure*, vol. 41, pp. 141–153, Oct. 2016.

[21] R. Q. Quiroga, H. Garcia, and A. Rabinowicz, "Frequency evolution during tonic-clonic seizures," *Electromyogr. Clin. Neurophysiol.*, vol. 42, no. 6, pp. 323–331, Sep. 2002.

[22] B. Hjorth, “EEG analysis based on time domain properties,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 29, no. 3, pp. 306–310, Sep. 1970.

[23] H. Hindarto, M. Hariadi, and M. Purnomo, “EEG signal identification based on root mean square and average power spectrum by using BackPropagation,” *J. Theor. Appl. Inf. Technol.*, Aug. 2014.

[24] N. Koolen *et al.*, “Line length as a robust method to detect high-activity events: Automated burst detection in premature EEG recordings,” *Clin. Neurophysiol.*, vol. 125, no. 10, pp. 1985–1994, Oct. 2014.

[25] M. Affinito, M. Carrozzi, A. Accardo, and F. Bouquet, “Use of the fractal dimension for the analysis of electroencephalographic time series,” *Biol. Cybern.*, vol. 77, no. 5, pp. 339–350, Nov. 1997.

[26] I. Farkas and E. Doran, “Activity Recognition from acceleration data collected with a tri-axial accelerometer,” *ACTA Tech. Napoc. Electron. Telecommun.*, vol. 52, no. 2, 2011.

[27] S. Iranmanesh and E. Rodriguez-Villegas, “An Ultralow-Power Sleep Spindle Detection System on Chip,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 11, no. 4, pp. 858–866, 2017.

[28] J. Novovičová, P. Somol, M. Haindl, and P. Pudil, “Conditional Mutual Information Based Feature Selection for Classification Task,” in *Progress in Pattern Recognition, Image Analysis and Applications*, vol. 4756, L. Rueda, D. Mery, and J. Kittler, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 417–426.

[29] J. Biesiada and W. Duch, “Feature Selection for High-Dimensional Data — A Pearson Redundancy Based Filter,” in *Computer Recognition Systems 2*, vol. 45, M. Kurzynski, E. Puchala, M. Wozniak, and A. Zolnierek, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 242–249.

[30] M. Dash, K. Choi, P. Scheuermann, and Huan Liu, “Feature selection for clustering - a filter solution,” in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, Maebashi City, Japan, 2002, pp. 115–122.

[31] R. Kohavi and G. H. John, “Wrappers for feature subset selection,” *Artif. Intell.*, vol. 97, no. 1–2, pp. 273–324, Dec. 1997.

[32] R. Battiti, “Using mutual information for selecting features in supervised neural net learning,” *IEEE Trans. Neural Netw.*, vol. 5, no. 4, pp. 537–550, Jul. 1994.

[33] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, “Conditional likelihood maximisation: a unifying framework for information theoretic feature selection,” *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 27–66, Feb. 2012.

[34] V. Hodge and J. Austin, “A Survey of Outlier Detection Methodologies,” *Artif. Intell. Rev.*, vol. 22, no. 2, pp. 85–126, Oct. 2004.

[35] “Outlier Detection,” Simon Fraser University.

[36] M. Delalandre, E. Valveny, T. Pridmore, and D. Karatzas, “Generation of synthetic documents for performance evaluation of symbol recognition & spotting systems,” *Int. J. Doc. Anal. Recognit. IJDAR*, vol. 13, no. 3, pp. 187–207, Sep. 2010.

[37] J. Lockman, R. S. Fisher, and D. M. Olson, “Detection of seizure-like movements using a wrist accelerometer,” *Epilepsy Behav.*, vol. 20, no. 4, pp. 638–641, Apr. 2011.

[38] J. Tian and H. Gu, “Anomaly detection combining one-class SVMs and particle swarm optimization algorithms,” *Nonlinear Dyn.*, vol. 61, no. 1–2, pp. 303–310, Jul. 2010.

[39] M. Nandan, S. S. Talathi, S. Myers, W. L. Ditto, P. P. Khargonekar, and P. R. Carney, “Support vector machines for seizure detection in an animal model of chronic epilepsy,” *J. Neural Eng.*, vol. 7, no. 3, p. 036001, Jun. 2010.

[40] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the fifth annual workshop on Computational learning theory - COLT ’92*, Pittsburgh, Pennsylvania, United States, 1992, pp. 144–152.

[41] M. de Berg, Ed., *Computational geometry: algorithms and applications*, 3rd ed. Berlin: Springer, 2008.

[42] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge, UK ; New York: Cambridge University Press, 2004.

[43] A. Ng, "Support Vector Machines." Stanford.

[44] G. Gordon and R. Tibshirani, "Lecture 16: October 18." UC Berkeley, 2012.

[45] J. Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," *Adv. KERNEL METHODS - SUPPORT VECTOR Learn.*, 1998.

[46] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the Support of a High-Dimensional Distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001.

[47] Y. Tang, "Deep Learning using Linear Support Vector Machines," *ArXiv13060239 Cs Stat*, Jun. 2013.

[48] C. A. Micchelli, Y. Xu, H. Zhang, and Gabor, "Universal Kernels," *J. Mach. Learn. Res.*, 2006.

[49] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New Support Vector Algorithms," *Neural Comput.*, vol. 12, no. 5, pp. 1207–1245, May 2000.

[50] P.-H. Chen, C.-J. Lin, and B. Schölkopf, "A tutorial on ν-support vector machines: ν-SUPPORT VECTOR MACHINES," *Appl. Stoch. Models Bus. Ind.*, vol. 21, no. 2, pp. 111–136, Mar. 2005.

[51] D. M. J. Tax and R. P. W. Duin, "Support Vector Data Description," *Mach. Learn.*, vol. 54, no. 1, pp. 45–66, Jan. 2004.

[52] A. Geletu, "Quadratic Programming Problems - A review on algorithms and applications (Active-set and interior point methods)."

[53] S. Sra, S. Nowozin, and S. J. Wright, Eds., *Optimization for machine learning*. Cambridge, Mass: MIT Press, 2012.

[54] G. C. Cawley and N. L. C. Talbot, "On Over-fitting in Model Selection and subsequent selection bias in performance evaluation," *J. Mach. Learn. Res.*, vol. 11, pp. 1079–2107, 2010.

[55] L. Bottou and C.-J. Lin, "Support Vector Machine Solvers," 2006.

[56] M. Claesen, F. De Smet, J. A. K. Suykens, and B. De Moor, “Fast Prediction with SVM Models Containing RBF Kernels,” *ArXiv14030736 Cs Stat*, Mar. 2014.

[57] C. K. I. Williams and M. Seeger, “Using the Nystrom method to speed up kernel machines,” *NIPS*, vol. 13, 2000.

[58] A. Rahimi and B. Recht, “Random Features for Large-Scale Kernel Machines,” *NIPS*, vol. 20, 2007.

[59] A. Menon, “Large-Scale Support Vector Machines: Algorithms and Theory,” 2009.

[60] X. Wu *et al.*, “Top 10 algorithms in data mining,” *Knowl. Inf. Syst.*, vol. 14, no. 1, pp. 1–37, Jan. 2008.

[61] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. 2009.

[62] B. A. Goldstein, A. E. Hubbard, A. Cutler, and L. F. Barcellos, “An application of Random Forests to a genome-wide association dataset: Methodological considerations & new findings,” *BMC Genet.*, vol. 11, no. 1, p. 49, Dec. 2010.

[63] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation Forest,” in *2008 Eighth IEEE International Conference on Data Mining*, Pisa, Italy, 2008, pp. 413–422.

[64] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation-Based Anomaly Detection,” *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 1, pp. 1–39, Mar. 2012.

[65] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, *Data structures and algorithms in Java*, Sixth edition. Hoboken, NJ: Wiley, 2014.

[66] B. Reed, “The height of a random binary search tree,” *J. ACM*, vol. 50, no. 3, pp. 306–332, May 2003.

[67] D. E. Knuth, *The art of computer programming*, 3rd ed. Reading, Mass: Addison-Wesley, 1997.

[68] S. Hariri, M. C. Kind, and R. J. Brunner, “Extended Isolation Forest,” *ArXiv181102141 Cs Stat*, Nov. 2018.

[69] J. Poland, “Three Different Algorithms for Generating Uniformly Distributed Random Points on the N-Sphere,” Oct. 2000.

[70] G. Regalia, F. Onorati, M. Lai, C. Caborni, and R. W. Picard, “Multimodal wrist-worn devices for seizure detection and advancing research: Focus on the Empatica wristbands,” *Epilepsy Res.*, vol. 153, pp. 79–82, Jul. 2019.

[71] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *ICML*, Atlanta, GA, USA, June 16, vol. 28.

[72] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[73] Z. C. Lipton, J. Berkowitz, and C. Elkan, “A Critical Review of Recurrent Neural Networks for Sequence Learning,” *ArXiv150600019 Cs*, May 2015.

[74] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, “Learning precise timing with lstm recurrent networks,” *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, Mar. 2003.

[75] K. Cho *et al.*, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” *ArXiv14061078 Cs Stat*, Jun. 2014.

[76] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *ArXiv150203167 Cs*, Feb. 2015.

[77] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How Does Batch Normalization Help Optimization?,” *ArXiv180511604 Cs Stat*, May 2018.

[78] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *ArXiv12070580 Cs*, Jul. 2012.

[79] S. Bai, J. Z. Kolter, and V. Koltun, “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling,” *ArXiv180301271 Cs*, Mar. 2018.

[80] A. van den Oord *et al.*, “WaveNet: A Generative Model for Raw Audio,” *ArXiv160903499 Cs*, Sep. 2016.

[81] F. Yu and V. Koltun, “Multi-Scale Context Aggregation by Dilated Convolutions,” *ArXiv151107122 Cs*, Nov. 2015.

[82] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *ArXiv151203385 Cs*, Dec. 2015.

[83] K. He and J. Sun, “Convolutional neural networks at constrained time cost,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015, pp. 5353–5360.

[84] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Highway Networks,” *ArXiv150500387 Cs*, May 2015.

[85] T. Salimans and D. P. Kingma, “Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks,” *ArXiv160207868 Cs*, Feb. 2016.

Appendix

Proof of XOR Uncorrelatedness

Consider an XOR function with the inputs $X_1, X_2 \sim Bernoulli(p)$, and output Y. Trivially, $E(X_1) = E(X_2) = p$. Then, $E(Y) = \sum y_i P(Y) = (0)(1-p)^2 + (0)(1-p)p + (1)(1-p)p + (1)p^2 = p^2 + p - p^2 = p$. To find $E(X_1 Y)$, we must first find the joint PDF $P(X_1, Y)$. This is done in the table below.

$X_1 = 0, Y = 0$	$X_1 = 1, Y = 0$	$X_1 = 0, Y = 1$	$X_1 = 1, Y = 1$
$(1-p)^2$	p^2	$p(1-p)$	$p(1-p)$

Then, $E(X_1 Y) = \sum P(X_1, Y) X_1 Y = p - p^2$. When we set $p = 0.5$, we get that $Corr(X_1, Y) = E(X_1 Y) - E(X_1)E(Y) = 0.25 - 0.25 = 0$, meaning they are uncorrelated for equal probabilities.

Proof of Mutual Information Formulation

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(x,y)$$

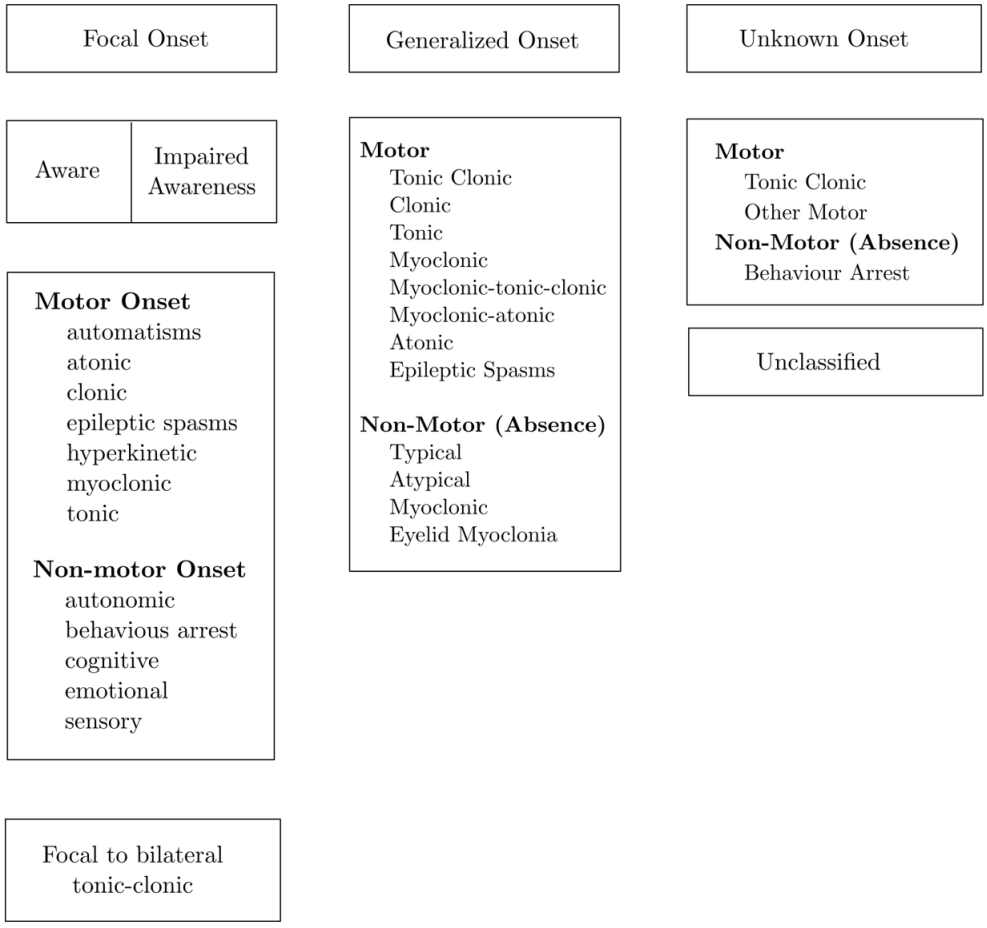
$$= -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(y|x)\, p(x)$$

$$= -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(x) - \sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(y|x)$$

$$= -\sum_{x \in X} p(x) \log p(x) - \sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(y|x)$$

$$= H(X) + H(Y|X)$$

Seizure Subtypes



Biography

Samyak Shah was born in 1995 in India. He did his undergraduate work in the University of Glasgow, where he majored in Biomedical Engineering. During his undergraduate studies, he spent a year in California, studying Biomedical Engineering at the University of California at Irvine. He spent three summer researching at InCube Labs in San Jose, and a summer interning at Philips Healthcare in San Diego. For his undergraduate thesis, he worked on the localization of ferromagnetic particles using GMR-based sensors under the guidance of Dr. Hadi Heidari. His paper "On-chip magnetoresistive sensors for detection and localization of paramagnetic particles" is a result of that work. In 2017, Samyak began his MSE in Biomedical Engineering with a concentration in Data Science at Johns Hopkins university. He joined Dr. Nathan Crone's lab to work on seizure detection using pattern recognition. During his time at Johns Hopkins, he was also a teaching assistant for the Biomedical Instrumentation course taught by Dr. Nitish Thakor.